AD-A040 675 BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS F/G 9/2
THE INTERFACE MESSAGE PROCESSOR PROGRAM.(U)
MAR 77 DAHC15-69-C-0179
UNCLASSIFIED TECHNICAL INFORMATION-89 NL

1 OF 2
AD
A040675

Bolt Beranek and Newman Inc.

2

14

Technical Information Report No. 89

6

THE INTERFACE MESSAGE PROCESSOR PROGRAM.

Originally published in February 1973

Update for IMP Version 3236

A025358

11

March 1977

12 119p.

15

Developed by Bolt Beranek and Newman Inc.
under Contracts No. DAHC15-69-C-0179, F08606-73-C-0027,
and F08606-75-C-0032, sponsored by the Advanced Research
Projects Agency, and Contract No. DCA200-C-616,
sponsored by the Defense Communications Agency

060 100

4B

TABLE OF CONTENTS

FIGURES

1. INTRODUCTION

The ARPA Network has been in operation for almost eight years and has become an international facility. The network has grown to more than sixty sites spread across the continental United States, plus satellite connections to Hawaii, Norway, and England, and is steadily growing; approximately one hundred independent computer systems of varying manufacture are interconnected. Provision has been made for terminal access to the network from sites which do not enjoy the ownership of an independent computer system. A map of the ARPA Network is shown in Figure 1-1.

Implementation of the IMPs required the development of a sophisticated computer program. This program has been previously described in [~~1,2~~]. As stated then, the principal function of the IMP program is the processing of packets, including the following: segmentation of Host messages into packets; receiving, routing, and transmitting store-and-forward packets; retransmitting unacknowledged packets; reassembling packets into messages for transmission into a Host; and generating RFNMs and other control messages. The program also monitors network status, gathers statistics, and performs on-line testing.

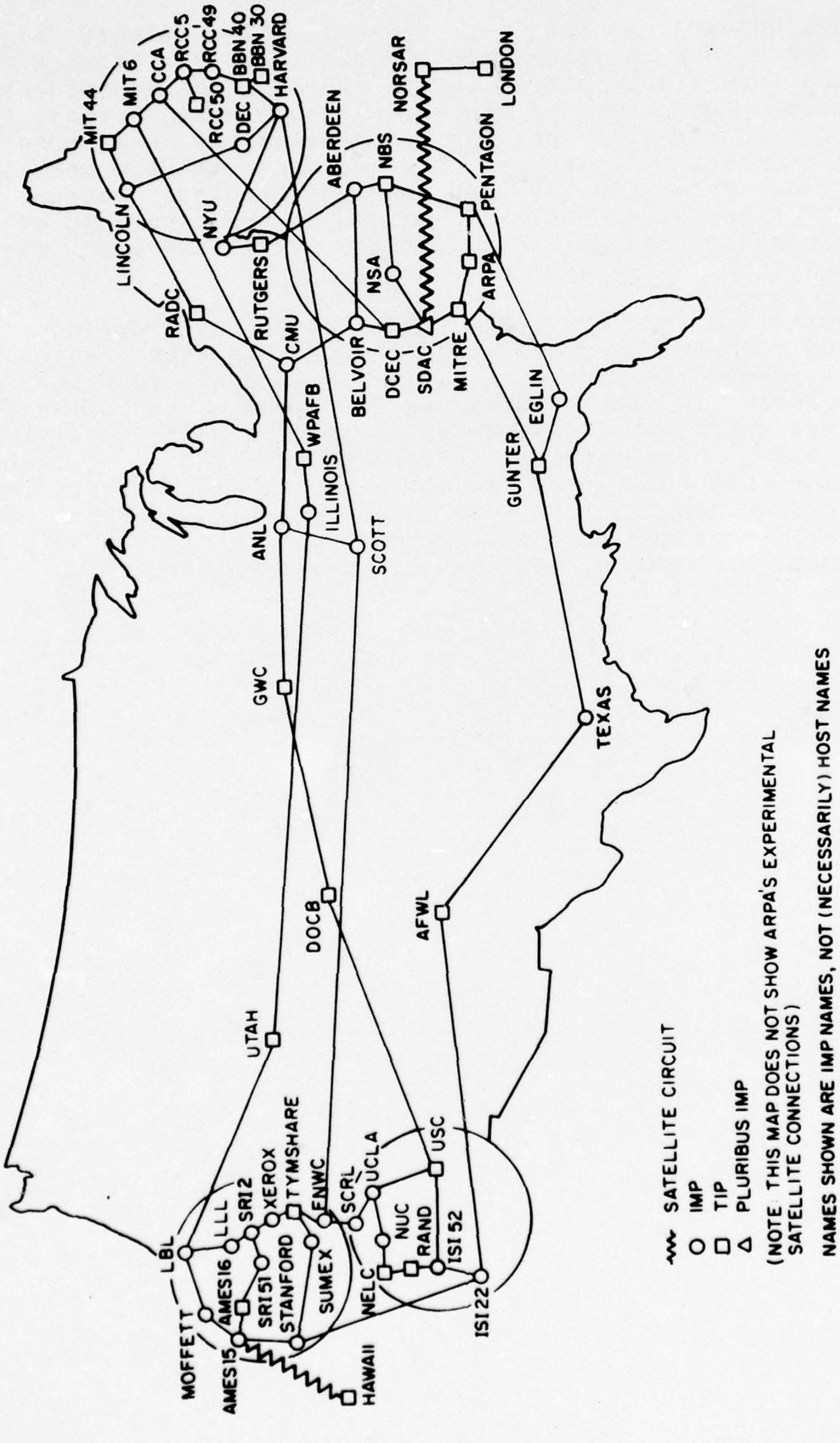


Figure 1-1 ARPANET Geographic Map,
March 1977

2. IMP PROCESSES

This section considers the algorithms that the IMP uses in performing its functions as a message-switching center and interface between Host computers. Figure 2-1 helps summarize some of the terms we will be using. The Host sends the IMP a message up to 8095 bits long. The message has a leader specifying its destination. The source IMP accepts the message in packets up to 1008 bits long. Each packet has a header to allow for the transmission from IMP to IMP. Figure 2-1 demonstrates how message 1 is transferred from IMP to IMP in these packets, numbered 1-1, 1-2, and 1-3. When a packet is successfully received at each IMP, an acknowledge or ack is sent back to the previous IMP. Inter-IMP acks are shown returning for each packet. Finally the message arrives at the destination IMP where it is reassembled: that is, the packets are recombined into the original message. The message is sent to the destination Host and when it has been accepted, a Ready for Next Message (RFNM) is sent back to the source Host. A RFNM is a unique, one-packet message and it is acknowledged. Several points are worth noting. First, acks are not actually separate transmissions, but are piggy-backed in returning packets to cut down on overhead. Next, packets on the inter-IMP lines are checksummed in both the modem interface hardware and the IMP software and the IMP employs a positive acknowledgement retransmission scheme. That is, if a packet is in error, it is not acknowledged. Then it is retransmitted until an acknowledge is received. Further, because of a dynamic routing, an IMP may send the several packets of a message out on different lines. For both of these reasons, the packets of a message may arrive at the destination IMP out of order and must be reassembled into the correct order for transmission to the destination Host.

2.1 IMP-Host Protocols

2.1.1 Messages and RFNMs

A major hazard in a message-switching network is congestion, which can arise either from system failures or from peak traffic flow. Congestion typically occurs when a destination IMP becomes flooded with incoming messages for its Host. If the flow of messages to this destination is not regulated, the congestion will back up into the network, affecting other IMPs and degrading or even completely clogging the communication service. To solve this problem a quenching scheme was developed that limits the flow of messages to a given destination before congestion begins to occur.

This quenching scheme consists of practices which allocate buffer space before a message may enter the system. If buffering is provided in the source IMP, one can optimize for low delay

transmissions. If the buffering is provided at the destination IMP, one can optimize for high bandwidth transmissions. To be consistent with the goal of a balanced communications system, an approach has been developed which utilizes some buffer storage at both the source and the destination; the solution also utilizes a request mechanism from source IMP to destination IMP.

Specifically, no multi-packet message is allowed to enter the network until storage for the message has been allocated at the destination IMP. As soon as the source IMP takes in the first packet of a multi-packet message, it sends a small control message to the destination IMP requesting that reassembly storage be reserved at the destination for this message. It does not take in further packets from the Host until it receives an allocation message in reply. The destination IMP queues the request and sends the allocation message to the source IMP when enough reassembly storage is free; at this point the source IMP sends the message to the destination.

Effective bandwidth is maximized for sequences of long messages by permitting all but the first message to bypass the request mechanism. When the message itself arrives at the destination, and the destination IMP is about to return the RFNM, the destination IMP waits until it has room for an additional multi-packet message. It then piggybacks a storage allocation on the RFNM. If the source Host is prompt in answering the RFNM with its next message, an allocation is ready and the message can be transmitted at once. If the source Host delays too long, or if the data transfer is complete, the source IMP returns the unused allocation to the destination. With this mechanism, the inter-message delay has been minimized and the Hosts can obtain the full bandwidth of the network.

The delay for a short message has been minimized by transmitting it to the destination immediately while keeping a copy in the source IMP. If there is space at the destination, it is accepted and passed on to a Host and a RFNM is returned; the source IMP discards the message when it receives the RFNM. If not, the message is discarded, a request for allocation is queued and, when space becomes available, the source IMP is notified that the message may now be retransmitted. Thus, no setup delay is incurred when storage is available at the destination.

These mechanisms make the IMP network fairly insensitive to unresponsive Hosts, since the source Host is effectively held to a transmission rate equal to the reception rate of the destination Host. Further, reassembly lockup is prevented because the destination IMP will never have to turn away a multipacket message destined for one of its Hosts; reassembly storage has been allocated for each such message in the network.

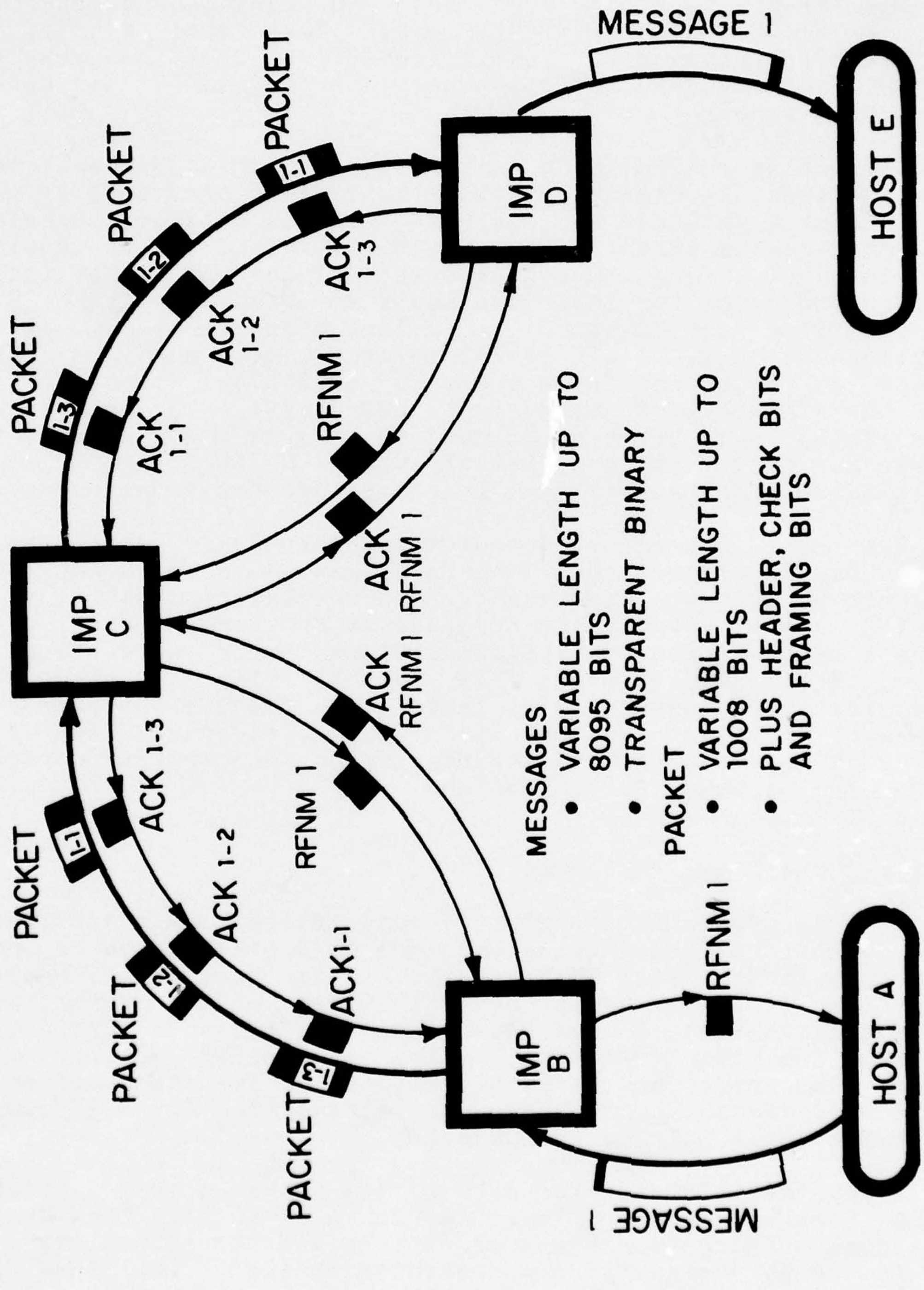


Figure 2-1 Message Protocol

2.1.2 Host-IMP Interfacing

Each IMP will service up to four Hosts whose cable distances from the IMP are less than 2000 feet. For distances greater than that, a modem channel must be used. This latter type of Host connection is termed a Very Distant Host (VDH). Procedures used for VDH connections are discussed in reference [3] and section 2.6 of this report.

Connecting an IMP to a wide variety of different local Hosts, however, requires a hardware interface, some part of which must be custom tailored to each Host. It was decided, therefore, to partition the interface such that a standard portion would be built into the IMP, and would be identical for all Hosts, while a special portion of the interface would be unique to each Host. The interface is designed to allow messages to flow in both directions at once. A bit-serial interface was designed partly because it required fewer lines for electrical interfacing and was, therefore, less expensive, and partly to accommodate conveniently the variety of word lengths in the different Host computers. The bit rate requirement on the Host line is sufficiently low that parallel transfers are not necessary.

The Host interface operates asynchronously, each data bit being passed across the interface via a Ready for Next Bit/There's Your Bit handshake procedure. This technique permits the bit rate to adjust to the rate of the slower member of the pair and allows necessary interruptions, when words must be stored into or retrieved from memory. The IMP introduces a preadjusted delay between bits that limits the maximum data rate; at present, this delay is set to 10 microseconds. Any delay introduced by the Host in the handshake procedure further slows the rate below this 100 Kbs maximum.

2.2 IMP-IMP Message Protocols

In order for a source Host to communicate with a destination Host, both source and destination IMPs must establish a record of the connection for that Host pair. This simplex connection, consisting of a Transmit Message Block at the source, and a corresponding Receive Message Block at the destination, is created, and later removed, using a special protocol which detects duplicate or missing messages. The connection is disallowed if the Host/Host access control mechanism does not permit that Host pair to communicate.

Every IMP maintains for each of its Hosts a pair of Host Access Control Words. These words are 16 bits long and the individual bits represent one of sixteen logical subnetworks; the bits in one word signifying membership in, and in the other word signifying permission to communicate with, the subnetworks. A pair of Hosts may communicate with each other only if they are

members of the same logical subnetwork or if one is allowed to communicate with Hosts in a subnetwork of which the other is a member.

Inter-Host messages start flowing as soon as the connection is positively established, or, if the connection is disallowed, the source Host is so notified. To insure that messages arrive at a destination Host in proper order, an independent message number sequence is maintained for each connection. A message number is assigned to each message at the source IMP and this message number is checked at the destination IMP. Out of an eight-bit message number space, both the source and destination keep a small window (currently eight) of valid message numbers, which allows several messages to be in flight simultaneously. Messages arriving at a destination IMP with message numbers outside of the current window or with message numbers already marked as received are duplicates to be discarded. The message number concept serves two purposes: it orders the messages for delivery to the destination Host, and it provides for the detection of duplicate and missing messages. The message number is internal to the IMP subnetwork and invisible to the Hosts.

A sequence control system based on a single source/destination connection, however, does not permit priority traffic to go ahead of other traffic. More generally, a Host may wish to specify a particular treatment for each message; thus, a separate connection is created for each "handling type". Currently, there are two possible handling types, regular (for high bandwidth) and priority (for low delay).

Since message numbers and reserved storage are so critical in the system, very stringent and careful procedures were developed to account for a lost message. The source IMP keeps track of all messages for which a RFNM has not yet been received, and the destination IMP keeps track of the replies it either has yet to send or has already sent. When the RFNM is not received for too long (presently about 30 seconds), the source IMP sends a control message (using the same message number) to the destination inquiring about the possibility of an incomplete transmission. Depending on the state of the reply table and message window at the destination, it will respond with either an indication that the message was not received or that it is out of range, or with a correct duplicate reply (RFNM). The source IMP continues inquiring until it receives a response. This technique generally insures that the source and destination IMPs keep their message number sequences synchronized and that any allocated space will be released should a message become lost in the subnetwork because of a machine failure.

A connection is terminated either after a prolonged period of inactivity (presently 3 minutes), or a somewhat shorter period of inactivity coupled with the need for the Message Block by some other connection, or by the need to resynchronize a message

number sequence that has been broken. The special termination protocol can be initiated by either the source or the destination in the first two cases above, or by the source in the third case, upon the receipt of an "out of range" response to an incomplete query. Upon closing a connection, both source and destination release all resources held or allocated for that connection.

There is a facility outside of the normal Host/Host connection mechanism for sending and receiving a stream of "raw packets". These messages are identified by a special Host-IMP and IMP-Host code and bypass the connection mechanism. They are routed normally through the subnetwork, but no sequencing, error control, reassembly, or storage allocation is performed. Thus, they may arrive out of order at the destination Host, some packets may be missing or duplicated, or packets may be thrown away by the subnetwork if insufficient resources are available to handle them. No RFNMs or other messages are sent back to the source Host about such raw packets.

2.3 IMP-to-IMP Channel Protocol

2.3.1 Logical Channel Protocol

A technique has been adopted for IMP-to-IMP transmission control which improves efficiency by 10-20% over the original separate acknowledge/timeout/ retransmission approach described in [1]. In the new scheme, which is also used for the Very Distant Host [3], each physical inter-IMP circuit is broken into a number of logical channels, currently eight in each direction. Acknowledgements are returned piggybacked on normal network traffic in a set of eight acknowledgement bits, one bit per channel, contained in every packet, thus requiring less bandwidth than the original method of sending each acknowledge in its own packet. In addition, the period between retransmissions is dependent upon the volume of new traffic. Under light loads the network has minimal retransmission delays, and the network automatically adjusts to minimize the interference of retransmissions with new traffic.

Each packet is assigned to an outgoing logical channel and carries the odd/even bit for its channel (which is used to detect duplicate packet transmissions), its channel number, and eight acknowledge bits - one for each channel in the reverse direction.

The transmitting IMP continually cycles through its used channels (those with packets associated with them), transmitting the packets along with the channel number and the associated odd/even bit. At the receiving IMP, if the odd/even bit of the received packet does not match the odd/even bit associated with the appropriate receive channel, the packet is accepted and the receive odd/even bit is complemented; otherwise the packet is a duplicate and is discarded.

Every packet arriving over a line contains acknowledges for all eight channels. The ack bits are set up at the distant IMP when it copies its receive odd/even bits into the positions reserved for the eight acknowledge bits in the control portion of every packet transmitted. In the absence of other traffic, the acknowledges are returned in null packets in which only the acknowledge bits contain relevant information (i.e., the channel number and odd/even bit are meaningless; null packets are not acknowledged). When an IMP receives a packet, it compares (bit by bit) the acknowledge bits against the transmit odd/even bits. For each match found, the corresponding channel is marked unused, the corresponding waiting packet buffer is discarded, and the transmit odd/even bit is complemented.

In view of the large number of channels, and the delay that is encountered on long lines, some packets may have to wait an inordinately long time for transmission. A one-character packet should not have to wait for several thousand-bit packets to be transmitted, multiplying by 10 or more the effective delay seen by the source. Therefore, the following transmission ordering scheme has been instituted: priority packets which have never been transmitted are sent first; next sent are any regular packets which have never been transmitted; finally, if there are no new packets to send, previously transmitted packets are periodically retransmitted even when there is a continuous stream of new traffic.

2.3.2 Physical Circuit Protocol

Each packet is individually routed from IMP to IMP through the network toward the destination. At each IMP along the way, the transmitting hardware generates initial and terminal framing characters and checksum digits that are shipped with the packet and are used for error detection by the receiving hardware of the next IMP. The format of a packet on an inter-IMP channel is shown in Figure 2-2.

Errors in transmission can affect a packet by destroying the framing and/or by modifying the data content. If the framing is disturbed in any way, the packet either will not be recognized or will be rejected by the receiver. In addition, the check digits provide protection against errors that affect only the data. The check digits can detect all patterns of four or fewer errors occurring within a packet, and any single error burst of a length less than twenty-four bits. An overwhelming majority of all other possible errors (all but about one in two to the twenty-fourth) is also detected. Thus, the mean time between undetected errors in the subnet should be on the order of years.

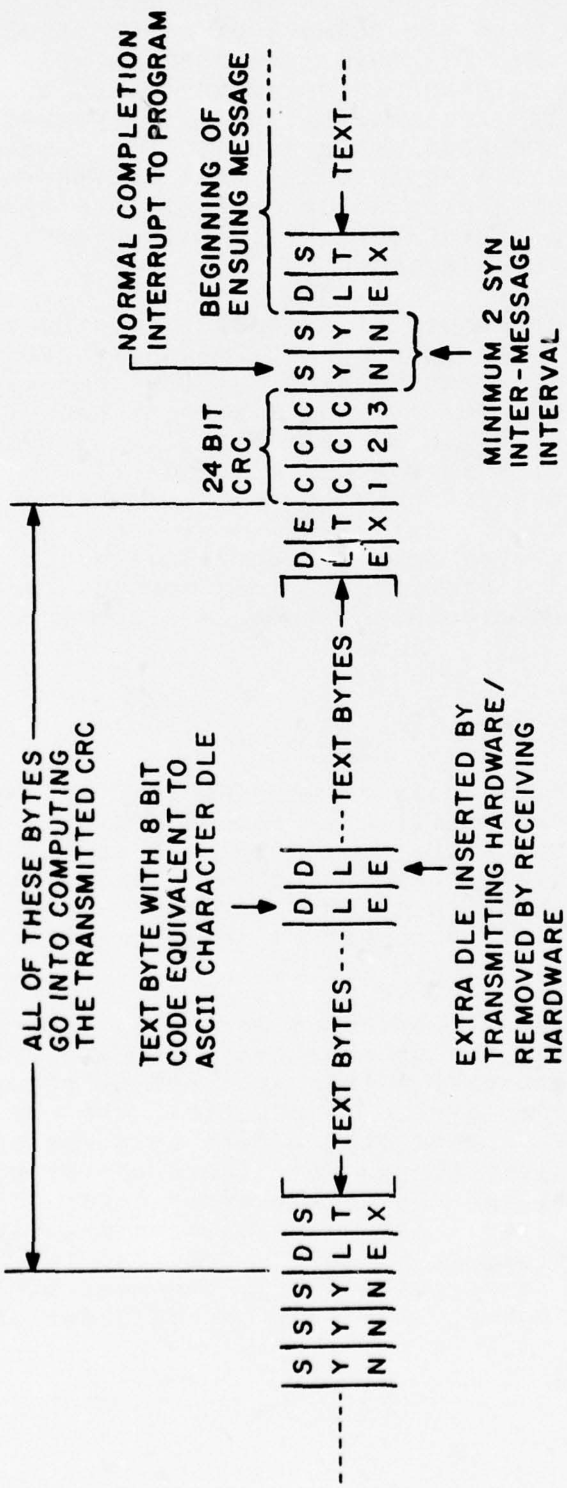


Figure 2-2 Packet Format on Line

2.4 Routing Algorithm

The routing algorithm directs each packet to its destination along a path for which the total estimated transit time is smallest. This path is not determined in advance. Instead, each IMP individually decides onto which of its output lines to transmit a packet addressed to another destination. This selection is made by a fast and simple table lookup procedure. For each possible destination, an entry in the table designates the appropriate next leg. These entries reflect line or IMP trouble, traffic congestion, and current local subnet connectivity. This routing table is updated whenever necessary, as described below.

Each IMP estimates the delay it expects a packet to encounter in reaching every possible destination over each of its output lines. It selects the minimum delay estimate for each destination and periodically passes these estimates to its immediate neighbors. Each IMP then constructs its own routing table by combining its neighbors' estimates with its own estimates of the delay to each neighbor. The estimated delay to each neighbor is based upon both queue lengths and the recent performance of the connecting communication circuit. For each destination, the table is then made to specify that selected output line for which the sum of the estimated delay to the neighbor plus the neighbor's delay to the destination is smallest.

The routing table is periodically and dynamically updated to adjust for changing conditions in the network. The system is adaptive to the ups and downs of lines, IMPs, and congestion; it does not require the IMP to know the topology of the network. In particular, an IMP need not even know the identity of its immediate neighbors. Thus, the leased circuits could be reconfigured to a new topology without requiring any changes to the IMPs.

The routing program has recently been modified to provide for more rapid and efficient propagation of routing messages. Through use of a technique called hold-down, the IMPs delay the route changeover process for a few seconds and in this way permit a faster and smoother cutover. When the best route is about to change, the IMPs first make sure that the neighboring IMPs know that the old route has gone bad before they attempt to change; this prevents the adjacent IMPs from slowing down the process by transmitting old information.

The IMPs also measure the bandwidth and loading of the circuits to which they are connected. The IMPs send routing proportionately more often on faster lines. In addition, they send routing proportionately more often on idle lines. Thus, the percentage of line bandwidth used for routing varies between 3% and 15%, approximately, as a function of line use.

Finally, the IMPs perform the routing computation on an incremental basis as each routing message is received. This means that the routing message being output on a given line is as up-to-date as possible. The routing messages carry serial numbers to permit the IMPs to detect that a new set of routing data has arrived which then is used, with the current data, to form the next routing message. There is a triple buffer pool for the routing message being output, the routing message being built, and the idle buffer, previously used for output. Each time a new input is received and no line is using the idle buffer for output, there is a cyclic permutation of the input, output, and free buffers.

Each of the routing messages sent and received carries a software checksum. In addition, the input, output, and timeout processes for routing all compute a checksum on the program before executing it, and a reload is initiated in the event of failure. These reliability measures are discussed in more detail in the next section.

2.5 Failure Protocols

The network is designed to be largely invulnerable to circuit or IMP failure as well as to outages for maintenance. Special status and test procedures are employed to help cope with various failures. In the normal course of events the IMP program transmits hellos (routing messages). The acknowledgement for a hello packet is an I-heard-you (IHY) bit in a returning null packet.

A dead line is detected by the sustained absence (approximately 3.2 sec) of IHY messages on that line. No regular packets will be routed onto a dead line, and any packets awaiting transmission will be rerouted. Routing tables in the network are adjusted automatically to reflect the loss. Receipt of consecutive I-heard-you packets for about 30 seconds is required before a dead line is defined to be alive once again. The IMP program takes into account the fact that an IMP may have only one working line to the network. In this case, a line may make twice as many errors as is usually permitted before it is declared unusable. This mechanism is an attempt to improve network availability from singly-connected sites.

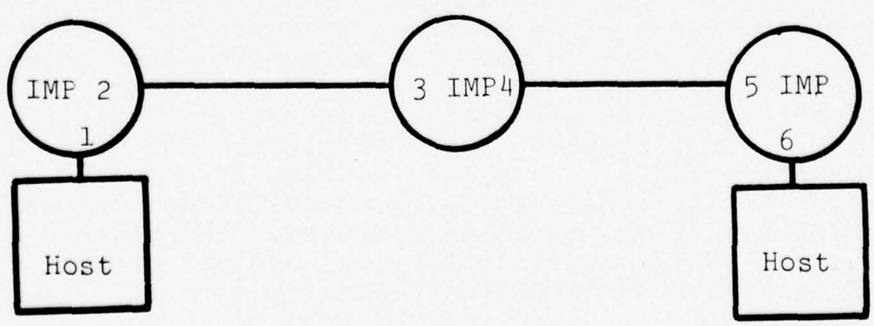
A dead line may reflect trouble either in the communication facilities or in the neighboring IMP itself. Normal line errors caused by dropouts, impulse noise, or other similar conditions should not result in a dead line, because such errors typically last only a few milliseconds, and only occasionally as long as a few tenths of a second. Therefore, it is expected that a line will be defined as dead only when serious trouble conditions occur.

If dead lines eliminate all routes between two IMPs, the IMPs are said to be disconnected and each of these IMPs will discard messages destined for the other. Disconnected IMPs cannot be rapidly detected from the delay estimates that arrive from neighboring IMPs. Consequently, additional information is transmitted between neighboring IMPs to help detect this condition. Each IMP transmits to its neighbors the length of the shortest existing path (i.e., number of IMPs) from itself to each destination. To the smallest such received number per destination, the IMP adds one. This incremented number is the length of the shortest path from that IMP to the destination. If the length ever exceeds the number of network nodes, the destination IMP is assumed to be unreachable and therefore disconnected.

Messages intended for dead Hosts (which are not the same as dead IMPs) cannot be delivered; therefore, these messages require special handling to avoid indefinite circulation in the network and spurious arrival at a later time. Such messages are purged from the network at the destination IMP. A Host computer is notified about another dead Host only when attempting to send a message to that Host.

The components of the IMP program dedicated to improving reliability have two main functions. First, the software is built to be as invulnerable as is possible in practice to hardware failures. Second, the software isolates and reports what failures it can detect to the NCC. With intermittent failures, it is important in practice to keep the IMP program running and diagnosing the problem rather than keeping the IMP down for long periods to run special hardware diagnostics.

The IMPs use the technique of software checksums on all transmissions to detect errors in packets, protecting the integrity of the data and isolating hardware failures. The end-to-end software checksum on packets, without any time gaps, works as follows:



-- A checksum is computed at the source IMP for each packet as it is received from the source Host (interface 1).

-- The checksum is verified at each intermediate IMP as it is received over the circuit from the previous IMP (interfaces 3 and 5).
-- If the checksum is in error, the packet is discarded, and the previous IMP retransmits the packet when it does not receive an acknowledgement (interfaces 2 and 4).
-- The previous IMP does not verify the checksum before the original transmission, to cut the number of checks in half. But when it must retransmit a packet it does verify the checksum. If it finds an error, it has detected an intra-IMP failure, and the packet is lost. If not, then the first transmission was lost due to an inter-IMP failure, a circuit error, or was simply refused by the adjacent IMP. The previous IMP holds a good copy of the packet, which it then retransmits (interfaces 2 and 4).
-- After the packet has successfully traversed several intermediate IMPs, it arrives at the destination IMP. The checksum is verified just before the packet is sent to the Host (interface 6).

This technique provides a checksum from the source IMP to the destination IMP on each packet, with no gaps in time when the packet is unchecked. Further, the length of each packet is verified. Any errors are reported to the NCC in full, with a copy of the packet in question. This method answers both requirements stated above: it makes the IMPs more reliable and fault-tolerant, and it provides a maximum of diagnostic information for use in fault isolation.

One of the major questions about such approaches is their efficiency. We have been able to include the software checksum on all packets without greatly increasing the processing overhead in the IMP. The method described above involves one checksum calculation at each IMP through which a packet travels. We developed a very fast checksum technique, which takes only one instruction execution per word. The program computes the number of words in a packet and then jumps to the appropriate entry in a chain of instructions. This produces a simple sum of the words in the packet, to which the number of words in the packet is added to detect missing or extra words of zero. With the inclusion of this code, the effective processor bandwidth of a Honeywell IMP is reduced by one-eighth for full-length store-and-forward packets. This add checksum is not a very good one in terms of its error-detecting capabilities, but it is as much as the IMP can afford to do in software. Furthermore, the primary goal of this modification is to assist in the remote diagnosis of intermittent hardware failures.

A different set of reliability measures has been instituted for routing. It is clear that catastrophic effects can follow for the network as a whole when a single IMP begins to propagate incorrect routing information. This failure may be due to a

memory failure in the data area or in the program itself. A single broken instruction in the part of the IMP program that builds the routing message causes the routing messages from the IMP to be random data. The neighboring IMPs interpret these messages as routing update information, and traffic flow through the network can be completely disrupted.

This kind of problem, the introduction of incorrect routing information into the network, can happen in three ways:

-- The routing message is changed in transmission. The inter-IMP checksum should catch this. The bad routing messages we saw in the Network had good checksums.
-- The routing message is changed as it is constructed, or the routing directory is changed as it is updated, say by a memory or processor failure, or before it is transmitted. This is what we termed above an intra-IMP failure.
-- The routing program is incorrect for hardware or software reasons.

The last two kinds of problems can be fixed by extending the concept of software checksums. The routing program has been modified to build a software checksum for the routing message as it builds the message, just as if it came from a Host. It is important that this checksum refer to the intended contents of the routing message, not the actual contents. That is, the program which generates the routing message builds its own software checksum as it proceeds, not by reading what has been stored in the routing message area, but by adding up the intended contents for each entry as it computes them. The process which sends out routing messages then always verifies the checksum before transmitting them. The routing directory checksum is computed once at initialization time. It is incrementally updated whenever an entry is changed, and it is verified every 640 ms. An error here causes a program reload. This scheme should detect all intra-IMP failures.

In addition, the routing program itself is checksummed to detect any changes in the code. The programs which copy in received routing messages, compute new routing tables, and send out routing messages each calculate the checksum of the code before executing it. If the program finds a discrepancy in the checksum of the program it is about to run, it immediately requests a program reload from an adjacent IMP. These checksums include the checksum computation itself, the routing program and any constants referenced. This modification should prevent a hardware failure at one IMP from affecting the network at large by stopping the IMP before it does any damage in terms of spreading bad routing.

2.6 Very Distant Host (VDH) Protocols

In instances where a Host is located more than 2000 feet from the IMP, connection is made by means of the standard modem interface hardware normally used for inter-IMP communication. Reference [3] contains a detailed description of the protocol used for this type of interface.

Briefly, the method used to assure successful IMP-Host transfers is similar to that used for the inter-IMP channels. Logical channels are used as described in section 2.3.1, although in this case only two channels are employed and the order of transmission is important. Therefore, both the Host and IMP software must be aware of packets. For example, assume packet A is transmitted from an IMP on channel 0, and packet B is then transmitted on channel 1. If an error were detected in packet A, but not B, no ack would be returned for A. The Host would retain packet B until A is retransmitted to it and received successfully, thus insuring delivery of the packets to its own processes in order A-B.

2.7 Packet Core Protocol

A special protocol is used for moving a portion of core memory from one IMP to another over the network. A sending or receiving process is implemented as either a Fake Host which is the ultimate recipient and generator of all messages, or a Fake Host together with a neighbor IMP operating in stand-alone (reloading) mode. In the latter case the Fake Host acts as port into the active network for the stand-alone IMP which is the ultimate recipient and generator of all messages. Also in the latter case, the live-to-stand-alone IMP communication is done via Packet Core messages, since the normal packet mechanisms are disabled in the stand-alone IMP.

There are only two message types in this protocol. One type is SETUP, which sets up internal variables that determine whether the process is sending or receiving, the location and size of the core transfer, and the network address of the foreign opposite process. The other type is CORE, which contains one or more segments of a core image. A packet core process which is idle is unlocked and neither sending nor receiving. An idle process will accept any CORE or SETUP message.

If a process accepts a "SETUP send" message, it locks to the foreign process specified in the SETUP data and begins to send core segments at a rate specified by the send/receive flag. After the last segment is sent, it resets to the idle state. A process that is sending will only accept messages from the foreign process it is locked to, and these may be either SETUP or CORE messages.

If a process accepts a "SETUP receive" message, it also locks to the foreign process, and waits for CORE messages. When a CORE message that completes the specified core transfer is received, the process is reset to the idle state. A process that is receiving will only accept messages from the foreign process it is locked to, and these may either be any SETUP, or a CORE message with a start address equal to the address the process is expecting next. If the address is too low (i.e. some previously received segment), the message is ignored. If it is too high (i.e. some segment was missed), a "SETUP send" message is sent to the foreign process, specifying the current address required. A "SETUP send" is also sent when no messages have been received for some time, and the process is reset to the idle state after a much longer period when no messages are received.

3. PROGRAM DESCRIPTIONS

Implementation of the IMPs required the development of a sophisticated operational computer program and the development of several auxiliary programs for hardware tests, program construction, and debugging. This section discusses the design of the operational program and describes the auxiliary software. Detailed program descriptions for the IMP software are included in Section 4.

3.1 General Descriptions

As previously mentioned, the principal function of the IMP operational program is the processing of packets. This processing includes segmentation of Host messages into packets for routing and transmission, building of headers, receiving, routing and transmitting of unacknowledged packets, reassembling of received packets into messages for transmission to the Host, and generating of RFNMs and acknowledgements. The program also monitors network status, gathers statistics, and performs on-line testing.

The entire program is composed of fifteen functionally distinct routines; each piece occupies no more than two or three pages of core (512 words per page). These routines communicate primarily through common registers residing in page zero of the machine which are directly addressable from all pages of memory. A typical map of core storage is shown in Figure 3-1. By use of a macro, code is "centered" on each physical page at assembly time such that there is an integral number of buffers between the last word of code on one page and the first word of code on the next. This technique eliminates all breakage (unusable core) except for part of one buffer on the very last page of core. Seven of the fifteen programs are directly involved in the flow of packets through the IMP: the task program performs the major portion of the packet processing, including the reassembly of Host messages; the modem programs (IMP-to-Modem and Modem-to-IMP) handle interrupts and the resetting of buffers for the Modem channels; the Host programs (IMP-to-Host and Host-to-IMP) handle interrupts and resetting of buffers for the Host channels, build packet headers during input, and construct allocation requests sent to the destination IMPs; the timeout program maintains a software clock, times out unused buffer allocations, reinitiates programs which have paused, and initiates routing computations and other relatively infrequent events. A background loop contains the remaining major programs and deals with initialization, debugging, testing, statistics gathering, and tracing. Background programs also initiate RFNM allocation and other sequencing and control messages. After a brief description of data structures, we will discuss packet processing in some detail.

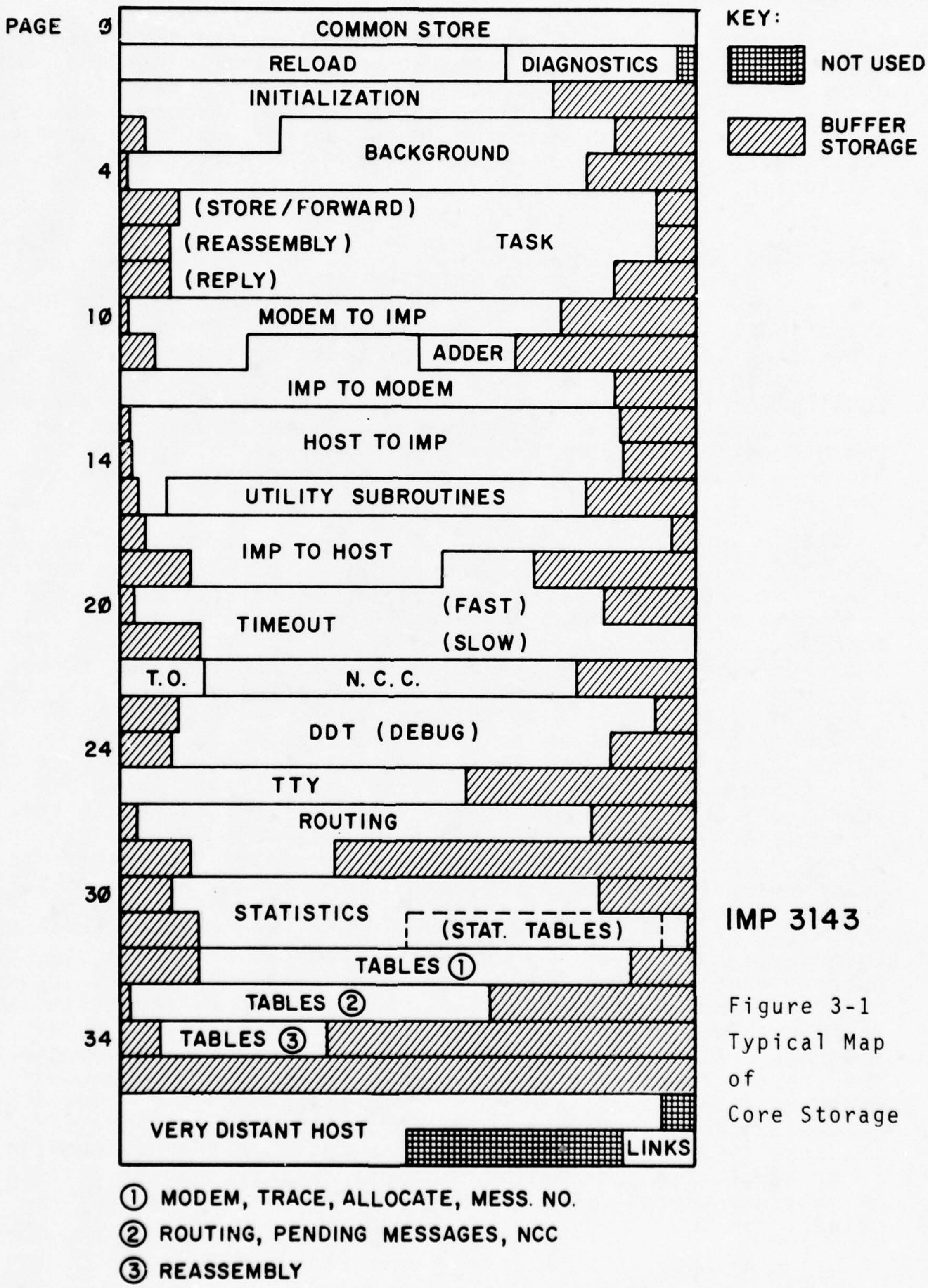


Figure 3-1
Typical Map
of
Core Storage

3.1.1 Data Structures

The major system data structures consist of buffers, queues and tables.

Buffer Storage. The buffer storage space consists of about 56 fixed length buffers, each of which is used for storing a single packet. An unused buffer is chained onto a free buffer queue and is removed from this list when it is needed to store an incoming packet. A packet, once stored in a buffer, is never moved. After a packet has been successfully passed along to its Host or to another IMP, its buffer is returned to the free list. The buffer space is partitioned in such a way that each process (store and forward traffic, Host traffic, etc.) is always guaranteed some buffers. For the sake of program speed and simplicity, no attempt is made to retrieve the space wasted by partially filled buffers.

In handling store and forward traffic, all processing is on a per-packet basis. Further, although traffic to and from Hosts is composed of messages, the IMP converts to dealing with packets; the Host transmits a message as a single unit but the IMP takes it one buffer at a time. As each buffer is filled, the program selects another buffer for input until the entire message has been provided for. These successive buffers will, in general, be scattered throughout the IMP's memory. An equivalent inverse process occurs on output to the Host after all packets of the message have arrived at the destination IMP. No attempt is made to collect the packets of a message into a contiguous portion of IMP memory. A typical allocation of buffer space in core storage is shown in Figure 3-1, as mentioned previously. IMPs with no Very Distant Host use the space on pages 35, 36 and 37 for buffer storage. All IMPs reserve the last 71 words for saving local data over reloads and for linkage to satellite and Terminal IMP programs.

The IMP program uses the following set of rules to allocate the available buffers to the various tasks requiring them:

- -- Each line must be able to get its share of buffers for input. Double buffering is provided for input on each line, which permits all input traffic to be examined by the program. Thus, acknowledgements can always be processed, which frees buffers.

- -- An attempt is made to provide enough store-and-forward buffers so that some lines may operate at full capacity. The number of buffers needed depends directly on line distance and line speed. The current limit is 1-1/2 times the number of channels per line, thus permitting 1-1/2 lines on the average to be operating at full capacity. Furthermore, each output line is guaranteed at least one buffer, thus permitting a low level of traffic on any line independent of congestion on other lines.

-- All remaining free buffers may be claimed for reassembly storage, including an overlap of nine into the store-and-forward allocation. All IMP processes (except for modem input) must share this storage using a priority scheme to resolve contention and preclude "deadly embrace" - type storage lockups.

Buffers currently in use are either dedicated to an incoming or outgoing packet, chained on a queue (or pointed to within a table) awaiting processing by the program, or being processed. Occasionally, a buffer may be simultaneously in several of these states, due to parallelism in the program. A "use count" is maintained within each buffer, and only when this count goes to zero is the buffer put back on the free queue.

Queues. There are three principal types of queues:

-- Task: All routing packets, all packets from the modems and all packets received on Host channels are placed on the task queue.

-- Output: A separate output queue is constructed for each inter-IMP modem circuit and each Host. Each modem output queue is subdivided into a priority queue and a regular message queue, which are serviced in that order. Each Host output queue is subdivided into a control message queue, a priority queue, and a regular message queue, which are also serviced in the indicated order.

-- Reassembly: The reassembly queue contains those packets being reassembled into messages for the Host.

Tables. Tables in core are identical for all IMPs, and their size is determined either by fixed IMP parameters or processing capability and network performance considerations. In the former category, there are per-IMP (67 word) tables for routing and statistics data; per-Host (8 word) tables for queue pointer, status, and local data required by re-entrant Host code; and per-line (5 word) tables similar in function to the Host tables. In the latter category, all tables are pooled resources which are acquired and relinquished by IMP processes depending on the activity required of them. These include transmit and receive message blocks, reassembly blocks, trace blocks, transaction blocks, and initialization data.

The size of the initialization code and the associated tables deserves mention. This was originally quite small. However, as the network has grown and the IMP's capabilities have been expanded, the amount of memory dedicated to initialization has steadily grown. This is mainly due to the fact that the IMPs are no longer identically configured. An IMP may be required to handle a Very Distant Host, or TIP hardware, or five lines and two Hosts, or four Hosts and three lines, or a very high speed

line, or a satellite link. As the physical permutations of the IMP have continued to increase, the criterion followed has been that the program should be identical in all IMPs, allowing an IMP to reload its program from a neighboring IMP and providing other considerable advantages. However, maintaining only one version of the program means that the program must rebuild itself during initialization to be the proper program to handle the particular physical configuration of the IMP. Furthermore, it must be able to turn itself back into its nominal form when it is reloaded into a neighbor. All of this takes tables and code. Unfortunately, the proliferation of IMP configurations which has taken place was not foreseen; therefore, the program differences currently cannot be conveniently computed from a simple configuration key. Instead, the configuration irregularities must be explicitly tabled.

3.1.2 Packet Flow Through Major IMP Routines

Figure 3-2 is a schematic drawing of packet processing. The processing programs are described below. Packet flow may be followed by referring to Figure 3-2.

The Host-to-IMP routine (H-I) handles messages being transmitted into the IMP from a local Host. The routine first accepts the leader to construct a header that is prefixed to each packet of the message. It then accepts the first packet and, if no allocation of space exists for the destination IMP, constructs a request for buffer allocation, which it places on the task queue. Single-packet messages are placed directly on the task queue regardless of allocation status and are held via a transaction block until either a RFNM or allocation is returned. A returned RFNM releases the packet. A returned allocation for the single-packet message will cause retransmission from the background loop. Requests for multipacket allocation are sent without actual message data. The request is recorded at the destination IMP and an allocation message is returned via the background loop when space is available. A returned allocation causes H-I to release the first packet with header to the task queue via the programmable task interrupt. Subsequent input is then accepted from the Host until end of message (EOM) occurs. The routine also tests a hardware trouble indicator and verifies the message format. The routine is serially reentrant and services all Hosts connected to the IMP.

The Modem-to-IMP routine (M-I) handles inputs from the modems. This routine first sets up a new input buffer, normally obtained from the free list. If a buffer cannot be obtained, the received buffer is not acknowledged and is reused immediately. The discarded packet will be retransmitted by the distant IMP. The routine processes returning acknowledgements for previously transmitted packets and either releases the packets to the free list or signals their subsequent release to the IMP-to-Modem

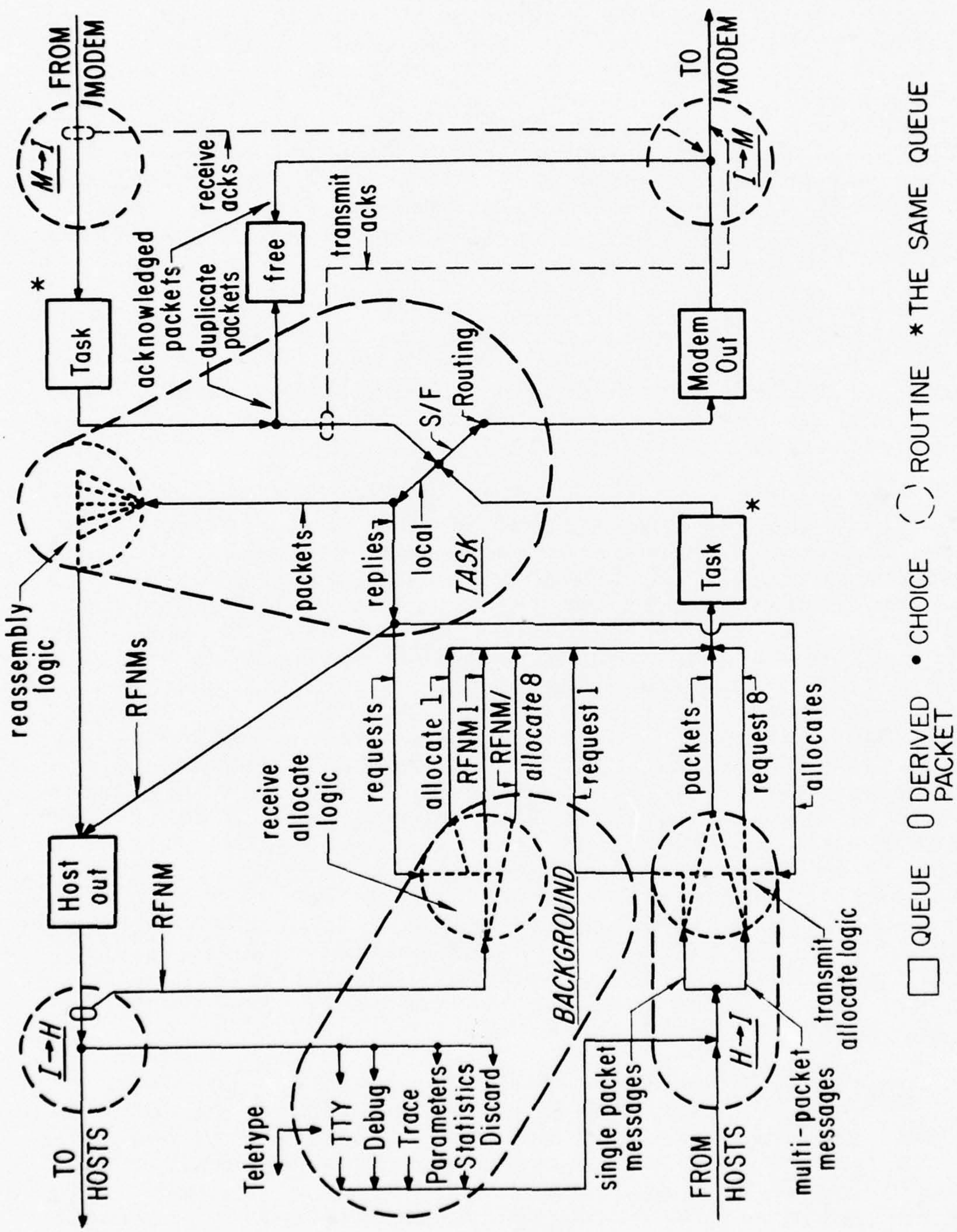


Figure 3-2 Packet Flow and Processing

routine. The M-I routine then places the buffer on the end of the task queue and triggers the programmable task interrupt.

The TASK routine uses the header information to direct packets to their proper destination. The routine is driven by the task interrupt, which is set whenever a packet is put on the task queue. The routine routes packets from the task queue onto an output modem or Host queue determined from the routing algorithm. If the packet is for non-local delivery, the routine determines whether sufficient store and forward buffer space is available. If not, buffers from modem lines are flushed and no subsequent acknowledgement is returned by the IMP-to-Modem routine. Normally, an acknowledgement is returned in the next outgoing packet over that modem line. Packets from Hosts which cannot get store and forward space are removed from the queue and replaced at a later time by the H-I routine.

If a packet from a modem line is addressed for local delivery, its message number is checked to see whether a duplicate packet has been received. As mentioned previously, each IMP maintains for each connection a window of contiguous numbers which it will accept from the other side of the connection. Packets with out-of-range numbers are considered duplicate and are discarded. The receipt of a RFNM for the oldest message by the source Host permits the window to be moved up by one number.

Replies such as RFNMs or Dead Host messages are placed in transaction blocks. TASK then pokes the IMP-to-Host routine to initiate output to the Host.

Message packets for local delivery are linked together with other packets of the same message number in a reassembly block. When a message is completely reassembled, the leading packet is linked to the appropriate Host output queue for processing by the IMP-to-Host.

Incoming routing messages are processed so that outgoing routing messages and the routing directory immediately reflect any new information received. The task routine generates I-heard-you messages in response as necessary to indicate to the neighbor receipt of the routing message.

IMP-to-Modem (I-M). This routine transmits successive packets from the modem output queues and sends piggybacked acknowledgements for packets correctly received by the Modem-to-IMP routine and accepted by the task routine.

IMP-to-Host (I-H). This routine passes messages to local Hosts and informs the background routine when a RFNM should be returned to the source Host.

Initialization and Background Loop. The IMP program starts in an initialization section that builds the initial data structures, prepares for inputs from modem and Host channels, and resets all program switches to their nominal state. The program then falls into the background loop, which is an endlessly repeated series of low-priority subroutines that are interrupted to handle normal traffic.

The programs in the IMP background loop perform a variety of functions: TTY is used to handle the IMP Teletype traffic; DEBUG, to inspect or change IMP core memory; TRACE, to transmit collected information about trace packets; STATISTICS, to take and transmit network and IMP statistics; PARAMETER-CHANGE, to alter the values of selected IMP parameters; PACKET CORE, to transfer portions of core images via the network; and DISCARD, to throw away packets. Selected Hosts and IMPs, particularly the Network Control Center, will find it necessary or useful to communicate with one or more of these background loop programs. So that these programs may send and receive messages from the network, they are treated as "fake Hosts." Rather than duplicating portions of the large IMP-to-Host and Host-to-IMP routines, the background loop programs are treated as if they were Hosts, and they can thereby utilize existing programs. The "For IMP" bit or the "From IMP" bit in the leader indicates that a given message is for or from a fake Host program in the IMP. Almost all of the background loop is devoted to running these programs.

The TTY program assembles characters from the Teletype into network messages and decodes network messages into characters for the Teletype. TTY's normal message destination is the DEBUG program at its own IMP; however, TTY can be made to communicate with any other IMP Teletype, any other IMP DEBUG program or any Host program with compatible format.

The DEBUG program permits the operational program to be inspected and changed. Although its normal message source is the TTY program at its own IMP, DEBUG will respond to a message of the correct format from any source. This program is normally inhibited from changing the operational IMP program; Network Control Center intervention is required to activate the program's full power.

The STATISTICS program collects measurements about network operation and periodically transmits them to a designated Host. This program sends but does not receive messages. STATISTICS has a mechanism for collecting measurements over 10-second intervals and for taking half-second snapshots of IMP queue lengths and routing tables. It can also generate artificial traffic to load the network.

The PACKET CORE program loads and dumps portions of its own IMP's core memory, or acts as an intermediary in loading and

dumping portions of the core memory belonging to a neighbor who is unable to communicate via the normal IMP-IMP protocol. The PACKET CORE facility allows for dissimilar machines to coexist as IMPs on the network; reloading and diagnostic dumping of a malfunctioning IMP can be done without the requirement that one of its neighbors be of the same machine type.

Other programs in the background loop drive local status lights and operate the parameter change routine. A thirty-two word parameter table controls the operation of the TRACE and STATISTICS programs and includes spares for expansion; the PARAMETER-CHANGE program accepts messages that change these parameters.

Other routines, which send connection protocol messages, send incomplete transmission messages, send allocations, return givebacks, send RFNMs, and retransmit single-packet messages also reside in the background program. These routines are called Back Hosts. However, these programs run in a slightly different manner than the fake Hosts in that they do not simulate the Host/IMP channel hardware. They do not go through the Host/IMP code at all, but rather put their messages directly on the task queue. Nonetheless, the principle is the same.

Timeout. The timeout routine is started every 25.6 ms (called a fast-tick timeout period) by a clock interrupt. The routine has two sections: the fast timeout routine which "wakes up" any Host or modem interrupt routine that has languished (for example, when the Host input routine could not immediately start a new input because of a shortage in buffer space); and the slow timeout routine which marks lines as alive or dead, updates the routing tables, and does long term garbage collection of queues and other data structures. (For example, it protects the system from the cumulative effect of such failures as a lost packet of a multiple packet message, where buffers are tied up in message reassembly).

These two routines, Fast and Slow, are executed so that fast timeout runs every clock tick (25.6 ms) and the slow timeout runs every 25th clock tick (640 ms). Although they run off a common interrupt, they are constructed to allow fast timeout to interrupt slow timeout should slow timeout not complete execution before the next timeout period. During garbage collection, every table, most queues, and many states of the program are timed out. Thus, if an entry remains in a table abnormally long or if a routine remains in a particular state for abnormally long, this entry or state is garbage-collected and the table or routine is returned to its initial or nominal state. In this way, abnormal conditions are not allowed to hang up the system indefinitely.

In addition to timing out various states of the program, the timeout routine is used to awaken routines which have put themselves to sleep for a specified period. Typically these

routines are waiting for some resource to become available, and are written as co-routines with the timeout routine. When they are restarted by Timeout the test is made for the availability of the resource, followed by another delay if the resource is not yet available.

3.1.3 Control Organization

It is characteristic of the IMP system that many of the main programs are entered both as subroutine calls from other programs and as interrupt calls from the hardware. The resulting control structure is shown in Figure 3-3. The programs are arranged in a priority order; control passes upward in the chain whenever a hardware interrupt occurs or the current program decides that the time has come to run a higher priority program, and control passes downward only when the higher priority programs are finished. No program may execute either itself or a lower priority program; however, a program may freely execute a higher priority program. This rule is similar to the usual rules concerning priority interrupt routines.

In one important case, however, control must pass from a higher priority program to a lower priority program - namely, from the several input routines to the task routine. For this special case, the computer hardware was modified to include a low-priority hardware interrupt that can be set by the program. When this interrupt has been honored (i.e., when all other interrupts have been serviced), the task routine is executed. Thus, control is directed where needed without violating the priority rules.

The practical implementation of priority control involves the setting of interrupt masks and enabling or inhibiting interrupts. Masks are built during initialization. In general, when a routine is entered, either by hardware- or software-initiated interrupt, the entering mask registers and keys are saved. A mask for the new routine is set into the mask register and the routine controls interrupts by executing INH or ENB commands. Therefore, H-I may inhibit interrupts by M-I for short periods of time during critical functions by using the INH. When the ENB command is executed, however, the mask bits for M-I will permit hardware interrupts transferring control from H-I. Interrupt control is obviously extremely critical and its use constitutes the most complex area of program operation.

All interrupt levels (except modem to IMP) make use of interrupt entry and exit routines that save the interrupted state variables on a stack. The six variables saved are: the index register, the accumulator, the keys (overflow bit, memory mode, etc.), the checksum routine return address, the interrupt mask, and the interrupt return address. The checksum routine is therefore re-entrant at each separate interrupt level and can be called with interrupts enabled. For a slight improvement in

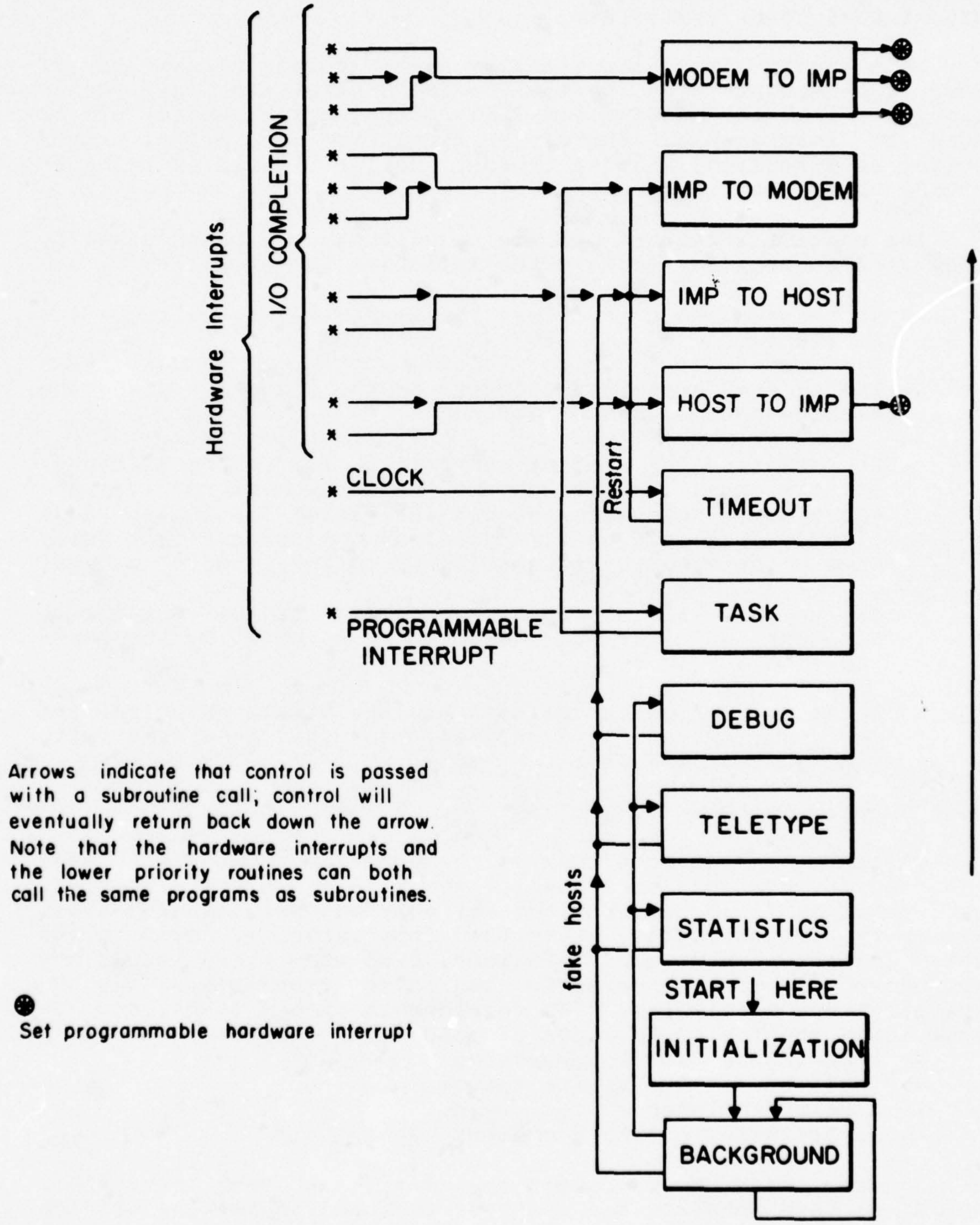


Figure 3-3 Program Control Structure

efficiency, the modem to IMP routine does its own saving and restoring of state variables.

Some routines must occasionally wait for long intervals of time, for example when the Host-to-IMP routine must wait for an allocation from the destination IMP. Stopping the whole system would be intolerable. Therefore, should the need arise, such a routine is dismissed, and the timeout routine will later transfer control to the waiting routine.

The control structure and the partition of responsibility among various programs achieve the following timing goals:

-- No program stops or delays the system while waiting for an event.

-- The program gracefully adjusts to the situation where the machine becomes compute-bound.

-- The Modem-to-IMP routine can deliver its current packet to the task routine before the next packet arrives and can always prepare for successive packet inputs on each line. This timing is critical because a slight delay here might require retransmission of the entire packet.

-- The program will almost always deliver packets waiting to be sent as fast as they can be accepted by the phone line.

-- Necessary periodic processes (in the timeout routine) are always permitted to run, and do not interfere with input-output processes.

3.1.4 Support Software

Designing a real-time program for a small computer with many high rate I/O channels is a specialized kind of software problem. The operational program required not only unusual techniques but also extra software tools; often the importance of such extra tools is not recognized. Further, even when these issues are recognized, the effort needed to construct such tools may be seriously underestimated. The development of the IMP system has resulted in the following kinds of supporting software:

-- Programs to test the hardware
-- Tools to help debug the system
-- A Host simulator
-- An efficient assembly process

So far, three hardware test programs have been developed. The first and largest is a complete program for testing all the special hardware features in the IMP. This program permits running of any or all of the modem interfaces in a crosspatched mode; it even permits operating several IMPs together in a test

mode. The second Hardware test runs a detailed phone line test that provides statistics on phone line errors. The final program simulates the modem interface check register whose complex behavior is otherwise difficult to predict.

The software debugging tools exist in two forms. Initially a simple stand-alone debugging program (DDT) was embedded into the operational program. This operational debugging program not only provides debugging assistance at a single location but may also be used in network testing and network debugging in a real-time environment.

The initial implementation of the IMP software took place without connecting to a true Host. To permit checkout of the Host-related portions of the operational program, a "Host simulator" was built that takes input from the console Teletype and feeds the Host routines exactly as though the input had originated in a real Host. Similarly, output messages for a destination Host are received by the simulator and typed out on the console Teletype. A program was also developed to automatically generate messages at a predefined rate and having predefined characteristics. The message generator routine is turned on, along with message rate, length and destination, via the parameter change fake Host program. The generator itself is embedded in the Statistics fake Host and is run during the background loop.

Without recourse to expensive additional peripherals, the assembly facilities on the DDP-516 are inadequate for a large program. (For example, a listing of the IMP program would require approximately 20 hours of Teletype output). Therefore, BBN uses other locally available facilities to assist in the assembly process. Specifically, a PDP-10 TENEX[4] text editor is used to compose and edit the programs, which are stored on a large random access file. The file is then used as a single source for a TENEX assembly program which assembles the IMP system, producing both object machine code and program listings.

The PDP-10 TENEX as a Host on the network provides several debugging aids. Core images of the current IMP system are created and maintained on the TENEX mass storage directly from assembled object code. When requested by the TENEX network utility program, an IMP's packet core transfer program dumps some or all of its core to the TENEX where it is verified against the core image and discrepancies are typed out. If an IMP has suffered a malfunction sufficient to cause control to pass to the stand-alone program, the TENEX utility program can be used to transfer the failed IMP's core image to the TENEX mass storage. The failed IMP can then be reloaded, minimizing the time it is out of service, while the dumped core image can be both verified and examined at some later time. If a diagnostic or corrective patch is required in a part (or all) of the network, it is made up as a small IMP assembly, and the utility program can quickly send the patch to the required IMP(s).

Additionally, fresh copies of Very Distant Host and TIP modules can be sent from the TENEX to a running IMP and started up using the utility program.

4. DETAILED PROGRAM DESCRIPTIONS

A concise, systematic approach has been taken in presenting some details of the IMP programs in the following pages. The approach is reflected by the headings of the outline used in describing the programs:

-- 1. Function - each function is numbered for reference in subsequent sections. The list of functions contains those which are fundamental to major IMP operations.

-- 2. Control Structure - A general description of the coding structure and its interrupt level.
   a. Entry points - locations and modes by which the program is entered.
   b. External calls - the names of subroutine or coroutine calls which the program makes.
   c. Initialization - important settings made during initialization process.
   d. Cleanup - actions taken before exiting or during unusual situations.

-- 3. Data Structures.
   a. Local data - variables and constants which are used only by the program.
   b. Shared - tables, variables and constants which are used by other programs as well. Care must be taken to use interrupt locks wisely so as to insure consistency in shared data.

-- 4. I/O Performed

PROGRAM DESCRIPTIONS

4.1 Modem to IMP

4.2 IMP to Modem

4.3 IMP to Host

4.4 Host to IMP

4.5 Timeout
- 4.5.1 Fast Timeout
- 4.5.2 Slow Timeout

4.6 TASK
- 4.6.1 TASK Store and Forward
- 4.6.2 TASK For Us

4.7 Background
- 4.7.1 SUCK
  - 4.7.1.1 TTY
  - 4.7.1.2 DDT
  - 4.7.1.3 Parameter Change and Packet Core
  - 4.7.1.4 Discard
- 4.7.2 JAM
  - 4.7.2.1 TTY
  - 4.7.2.2 DDT
  - 4.7.2.3 Trace and Packet Core
  - 4.7.2.4 Statistics
- 4.7.3 Back Hosts

4.8 Very Distant Host (VDH)
- 4.8.1 VDH Initialization
- 4.8.2 VDH Input Interrupt
- 4.8.3 VDH Output Interrupt
- 4.8.4 VDH Timeout
- 4.8.5 VDH Background

4.9 Initialization

4.10 Miscellaneous
- 4.10.1 Power Failure
- 4.10.2 Watchdog Timer
- 4.10.3 Stand-alone

PROGRAM DESCRIPTION 4.1

NAME

Modem to IMP

FUNCTION

1) Process input interrupts and initiate new modem inputs.
2) Verify packet checksums and lengths.
3) Pass packets to TASK.
4) Free acknowledged packets.

4a) Trace acknowledged packets if necessary. Copy send and acknowledge times from packet to trace block. Mark the output channel and complete indicators.

5) Put blocks on the block queue.

CONTROL STRUCTURE

Function 1 is performed by straight line code which is duplicated for each modem. Function 2 is performed by the shared add chain. Functions 3, 4 and 5 are performed by shared code. Function 4a is performed by the subroutine TRCDUN which is also called by IMP to Host. Only one modem runs at a time. The entire routine runs with interrupts disabled.

ENTRY POINTS

Modem to IMP hardware interrupts come to M2In where n is modem number 1-5. These are the only entrances. No software calls are made.

EXTERNAL CALLS

There are no direct calls. The TASK interrupt is forced by the OCP TASK instruction.

INITIALIZATION

The first input on each modem after initialization is discarded to avoid devoting a buffer to input on an unused modem.

CLEANUP

When a line goes down, input is turned off for a specified time by KILLIN and reinstated in Timeout by DEDL.

DATA STRUCTURES

LOCAL DATA

TA, TX, TK, and TAR are the save registers for the A, X registers, the keys, and the checksum routine return. The priority interrupt mask is not changed. The active modem number is saved in MP.

SHARED DATA

SFQ, EFQ: Accessed to obtain a buffer for the next input, and also to free acknowledged packets.

STQ, ETQ: Packets placed on the queue for TASK processing.
I2MTAB: Acknowledged packets freed.
I2MNXT: The packet marked as currently being sent out a particular line is not freed immediately when acknowledged, but a flag is set for modem output to perform this function.
TSEX, CHFREE: Acknowledgement bits expected on channels in use.
SPCQ, EPCQ: Packet core messages are placed on the queue for background processing.

I/O PERFORMED

Input of header and data into a packet buffer, which is later identified as a data packet, routing message, packet core message or null packet.

PROGRAM DESCRIPTION 4.2

NAME

IMP to Modem

FUNCTION

1) Process output interrupts.
2) Free the packet last sent if acknowledged while being sent.
3) Send routing messages, core loads, reload demands, and packet core messages.
4) Retransmit packets unacknowledged for 125 ms. verifying their checksums and lengths.
5) Send new priority packets.
6) Send new regular packets.
7) Send null packets of acknowledgements only.

CONTROL STRUCTURE

The routine performs functions 1 and 2 and then attempts to perform functions 3 through 7 in that order, as required. If none are required, the modem output is marked inactive for that channel. Only modem input interrupts are enabled. Checksum verification is performed using the shared add chain.

ENTRY POINTS

IMP to Modem hardware interrupts come to I2Mn where n is the modem number 1-5. Software interrupts come to I2MSB whenever any traffic is generated for an inactive modem. A periodic wakeup of IMP to Modem from Timeout is also necessary to perform retransmissions in the absence of other traffic.

EXTERNAL CALLS

None.

INITIALIZATION

Each modem is initialized to be down (see next section).

CLEANUP

When a line goes down, output is turned off for a specified time, then all queues and tables are garbage-collected and any pending packets are marked for retransmission on other lines. Then output is reinstated. There is also a timeout on outputs which are not completed within 30 seconds.

DATA STRUCTURES

LOCAL DATA

The active modem number is saved in OCHN. A pointer into I2MTAB is kept for retransmissions.

SHARED DATA

1) NONE: Busy flag =0 idle, <0 busy.
2) I2MNXT: Pointer to last packet output, flagged if acknowledged while being output.
   SFQ, EFQ: Pointers to free buffer queue.
3) SLT: Flag to send a line test (routing message) or a reload demand, or to block output.
4) I2MTAB: Table of NACH slots per line, containing packet pointers.
5) SMPQ, EMPQ: Priority modem output queue.
6) SMQ, EMQ: Regular modem output queue.
7) SNULL: Flag to send a null packet of acks only.

I/O PERFORMED

3) Output of routing table from fixed core area, or packet core message from buffer, common to all modems.

4,5,6) Output of header and data from packet buffer.

7) Output of null packet from fixed core area, one per modem.

PROGRAM DESCRIPTION 4.3

NAME

IMP to Host

FUNCTION

1) Process output interrupts.
2) Mark to send RFNM and allocate if just transmitted first packet.
3) Free the packet just sent.
4) Trace packet just sent if necessary.
5) Verify checksum and length and send next packet in message.
6) Send next control message.
7) Send next priority message.
8) Send next regular message.

CONTROL STRUCTURE

The routine performs function 1, checks to see if 2 is required, and performs function 3. Then functions 4-8 are performed if necessary. If none are, no action is taken. Modem input and output interrupts are enabled. Checksum verification is performed by the checksum routine.

ENTRY POINTS

IMP to Host hardware interrupts come to IHnE, where n is the host number (0-3). Software interrupts come to IHSB whenever traffic is generated for an inactive host, whenever Timeout has ticked over a host's 'alarm clock', or whenever a fake host has finished with a packet. The TIP's software calls IMP to Host via its associated hardware entry (currently IH2E), as do the VDH hosts.

EXTERNAL CALLS

2) RALLYP: Enters RFNM into receive message block for use by Background; also called by Background and Task.
3) FLUSH: Returns a buffer to the free queue.
4) TRCDUN: Traces a packet; also called by Modem to IMP.
5) HTPPF: Counts a packet of throughput for trouble reports.
7)-8) HTPMF: Counts a message of throughput.

INITIALIZATION

All real hosts are initialized to be held down for a specific delay (30 seconds), to empty all their queues, and to send 3 NOPs. This action is begun before Background runs.

CLEANUP

All control entries are cleared and all packets on a host's priority and regular queues are freed, the

host's ready line is flapped, and an 'Interface Reset' message returned to the host, if:

a) A control message is in transmission for more than a specified time, or

b) Any message is on a host queue for more than that specified time.

That maximum is currently set to 30 seconds.

DATA STRUCTURES

LOCAL DATA

IHP contains the currently running host number. Tables indexed by host include:

IHLO: Saved co-routine reentry points.
IHSP: Saved buffer pointers.
IHWQ: Saved host queue pointers.
IHLSTP: Last-packet flag.

SHARED DATA

1) IHTT: Timeout alarm clock, used to wake up IMP to Host if a packet is too long in transmission.
3) SFQ, EFQ: Queue of free buffers.
5) HTPMFL, HTPMFN: Message throughput counters.
6) IHLFLG: Host control message bits for internally generated control messages.
SHRQ, EHRQ: Host control message queue.
7) SHPQ, EHPQ: Host priority queue.
8) SHQ, EHQ: Host regular message queue.
7)-8) HTPPFL, HTPPFN: Packet throughput counters.

Also, cleanup and initialization use HINWAT to inform Host to IMP that the host is to be held off.

I/O PERFORMED

5)-8) OCP to initiate a middle-output or a final-output is issued.

Also, cleanup and initialization enable and disable the host interface.

PROGRAM DESCRIPTION 4.4

NAME

Host to IMP

FUNCTION

1) Process input interrupt.
2) For input of control message, take appropriate action, and initiate input.
3) For input of regular message leader, begin processing of message, and initiate input of first packet of message.
4) For input of first packet, if destination has not allocated space, initiate request.
5) For input of first packet, if destination has allocated space, process packet and initiate input of subsequent packet. For all packets, check length and generate software checksum.
6) For input of subsequent packet, process the packet and initiate input of subsequent packet.

CONTROL STRUCTURE

Modem input and output and host output interrupts are enabled. Function 1 is performed, then one of the remaining functions, with processing resuming from the last co-routine exit. Processing of packets and requests for allocation includes generation of checksum using the checksum routine.

ENTRY POINTS

Host input hardware interrupts come to HInE, where n is the host number (0-3). Software interrupts come to HISB whenever Timeout has ticked over a host's 'alarm clock', when Task has processed a reply, or when a fake host has a packet to send. The TIP's software calls Host to IMP via its associated hardware entry (currently HI2E), as do the VDH hosts.

EXTERNAL CALLS

FLUSH: Returns a packet to the free queue.
BLKSRC: Find a message block; also called by Task.
MESGET: Assigns a message number; also called by Background.
TSBPUT: Make entry in Transaction Block Table.
TSBGET: Get free transaction block for control message.
GETALL: Retrieves transmit allocate from message block, if any; also called by Task and Background.
HTPMT: Counts a message of throughput for trouble report.
HTPPT: Counts a packet of throughput.

INITIALIZATION

Initialization is accomplished by setting HILO, the co-routine entry points and EMFH, the end test instructions. Real hosts start by discarding the initial input. Fake hosts start by expecting input of a leader. Initial input is blocked according to HINWAT, which is controlled by IMP to Host and Timeout.

CLEANUP

A host is initialized as above when an error is detected, when a host has been down, or when an input takes too long (currently when over 15 seconds elapses while waiting for input subsequent to the leader).

DATA STRUCTURES

LOCAL DATA

HIP saves the number of the calling host. Tables indexed by host include:
HISP: Saved buffer pointers.
HILO: Saved co-routine entry points.
EMFH: Host's end of message test instruction.
HILINK: Pointer to saved link or sub-code.
HIHT, HIHH, HIHS, HIHP, HIHI, HIHL: Used for building headers.
HIBL: Current message block pointer.
HITYPE: Current packet type (raw, multi-, or single-packet).
HIWTIM: Wait timer.

SHARED DATA

HITT: Timeout alarm clock.
HINWAT: Flag from IMP to Host to block input.
SFQ, EFQ: Queue of free buffers.
RUT: Route Use Table, used to see if an IMP is up.
ETQ: Task queue.
TSBTAB: Transaction Block Table.
HIALLT: Host allocate timer and count.
HTPMTL, HTPMTN, HTPPTL, HTPPTN: Message and packet throughput counters.
HDOWN: Table of host-going-down statuses.
EHRQ: Host control message queue.
HSTMAP: Physical/logical Host map.
HILTYP: Old/new leader style flag (-1 old, 0-9 = padding for new style).

I/O PERFORMED

OCP's to initiate input; SKS's to test for errors and end of message; and input from the Real time clock.

PROGRAM DESCRIPTION 4.5

NAME

Timeout

FUNCTION

1) Service the 25.6 ms clock interrupt and count time in 25.6 ms units.
2) Call JOBF, Fast Timeout, 24 out of 25 ticks.
3) Call JOBS, Slow Timeout, 1 out of 25 ticks.

CONTROL STRUCTURE

The Slow Timeout routine enables clock interrupts. That is, it is constructed to be able to interrupt itself and keeps track of which routine to run at the next clock tick. In this way, Fast Timeout may interrupt Slow Timeout. All Modem and Host input and output interrupts are also enabled. The Slow Timeout routine falls into Fast Timeout, which thus in effect runs every tick.

ENTRY POINTS

Hardware 25.6 ms clock interrupt only.

EXTERNAL CALLS

None.

INITIALIZATION

The first Timeout is initialized to be a Slow Timeout tick. All Timeout programs run before background runs for the first time.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

TOSLOW: Slow Timeout counter.

SHARED DATA

None.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.5.1

NAME

Fast Timeout

FUNCTION

1) RSTSWP: Rotate output, working, idle routing buffers if necessary and possible.
2) RSTOUT: Send routing, determine line status.
3) IMTC: Periodic wakeup of IMP to Modem.
4) HITC: Periodic wakeup of Host to IMP.
5) SWCH: Monitor flags and switches which generate immediate NCC reports.

CONTROL STRUCTURE

A series of subroutine calls, one for each function.

ENTRY POINTS

Called directly by Timeout 24 of every 25 ticks and by Slow Timeout 1 of every 25. Thus it runs every 25.6 ms.

EXTERNAL CALLS

2) DEDL: Timing and routing on a dead line.
   KILLIN (from DEDL): Kill a line.
   RSTINP (from DEDL): Process a (dummy) routing input.
3) I2MSB: Software interrupt of IMP to Modem.
4) HISB: Software interrupt of Host to IMP.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

2) RSTDT: Temporary index register.
   RMCLKS: Table of clock counters for line speeds.
   RMBIT: Table of routing flag counters, one per line.
   LTR: Line going down counter.
   SENR: DEDL Temporary.
   JSRTQ: DEDL Temporary.
3) IMTK: Loop counter to attempt to wake up each modem output in numerical order.
4) HITK: Loop counter to attempt to wake up each host input in random order.

SHARED DATA

1) RST.O, RST.F, RST.N: Pointers to output, free, and next routing buffers.
2) SLT: Line test/dead flag.
   E123: Line error counter.

LAC: Line alive counter.
LINE: Line alive dead status.
NEIGHB: Line's neighbor IMP number.
LEND: Low numbered end bit.
RTRCVD: Line tests received count.
RTSSNT: Line tests sent count.
I2MTAB: Table of modem output pointer slots.
ERRQ: Reroute queue.
TSEX, RSEX: Acknowledge bits.

3) NONE: An alarm clock for IMP to Modem to indicate whether each modem is waiting for hardware interrupts (<0 and a timer) or idle (=0) and thus requires software wakeup.
4) HITT: A similar alarm clock for Host to IMP. If Host to IMP is waiting for a resource or timing out an input, it uses HITT as its alarm clock.
   HIALLT: Allocate timeout.
5) Flags which are monitored for immediate NCC reports.
   RSFNCC: A restart/reload/wdt/power fail indicator.
   HLTLOC: The address of the last pseudo-halt.
   HLNM: The number of the host interface under test.
   TRON, SNON, SON, MGON: the status of various statistics programs.
   Also, the state of memory protect and sense switches.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.5.2

NAME

Slow Timeout

FUNCTION

1) IHTC: Periodic wakeup of IMP to Host.
2) DEDH: Host alive/dead status check.
3) RUTCLK: Periodic routing functions.
4) HTEST: Perform interface loop/unloop/clear.
5) HPOKE: Test host interfaces with data.
6) RESETT: Clean up transmit structures and data to dead IMPs, time out transmit message block functions.
7) RESETR: Clean up receive structures and data from dead IMPs, time out receive message block functions.
8) JUQC: Update queue counters.
9) BUFCHK: Check buffer management flags.
10) Verify that add chain is intact.
11) VDH.TO: VDH Timeout call.
12) Count up the Software Watchdog Timer.

CONTROL STRUCTURE

A series of subroutine calls, one for each function, except (10) and (12), which are in line.

ENTRY POINTS

Called by Timeout 1 out of every 25 clock ticks, or every 640 ms.

EXTERNAL CALLS

1) IHSB: Periodic wakeup of IMP to Host.
2) IHST: IMP to Host is restarted if the Host is detected to be hardware-down.
6) TSBGET: Find Transaction Block for pending message.
   OWP: Put one word message.
7) REASF: Free reassembly block and packets; also called by Task.
12) SWDTIL: Reloads the IMP (called by a faked interrupt), if Software Watchdog Timer is not reset for 3 minutes.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

1) IHTK: Loop counter.
2) DHC: Loop counter.
4) HTOLD: Saved HTPAR flag.
   HTINTF: Interface under test.
6,7) RSTIMP: IMP to reset, -1 if none.

RSTBLK: Message block to notify.
RSTCNT: Temp. counter.
RSTPTR: Temp. pointer.
RSTCOD: IMP-host code to send.
RSTSBC: IMP-host subcode to send.
RSTCHT: Saved X register.
RSTPCT: Pointer to queue counters.
RSTPQT: Pointer to queue end pointers.
RSTIHQ: Saved IHWQ.
COMHST: Local Host.
COMBLK: Current block pointer.
MBTFRE: Temp free block count.
MBTCNT: Temp message block count.
TIMCNT: Five timers counting varying amounts of slow ticks.

9) BUFREQ: Flag to initiate removal or return of a buffer.

SHARED DATA

1) IHTT: Alarm clock for IMP to Host.
5) HTPAR: Function and interface test command.
6) TSBTAB: Transaction Block Table.
   HIRST: Flag to IMP-to-Host that message is from reset IMP.
   B1FCNT: Free transmit block count.
7) REASTB: Reassembly Block Table.
   MESSTK: Completed message stack.
   SHQ, SHPQ: host regular and priority queues.
   IHWQ: Host pointers to current queues.
   IHLSTP: IMP-to-host last packet flags.
   B2FCNT: Free receive block count.
8) All queue counters.
9) BUFFLU - Flag for FLUSH to steal a buffer.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.6

NAME

TASK

FUNCTION

1) Process interrupts and service the TASK queue.
2) Perform duplicate detection if modem input.
3) For routing input, copy information and monitor line up/down status.
4) For packet input, branch to either TASK Store-and-forward or TASK For Us.
5) Return an ACK or a NACK.

CONTROL STRUCTURE

TASK has the logical structure of a subroutine called by Modem or Host input, which provides two returns: input accepted and input rejected. It is implemented as an interrupt routine, called by a settable interrupt. It processes all inputs on the TASK queue and then dismisses. All interrupts except TASK are enabled.

ENTRY POINTS

Called by Modem to IMP, Host to IMP, and Background, by programmed interrupt.

EXTERNAL CALLS

5) I2MSB: Software interrupt of IMP to Modem if necessary to return acknowledgement.
   HISB: Software interrupt of Host to IMP to communicate acceptance or rejection of the input.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

THIS: Pointer to the packet TASK is processing.

2) ACKP, ACKBIT: Pointer to RSEX and bit pointer corresponding to input channel.

SHARED DATA

1) STQ, ETQ: TASK queue.
2) RSEX: Receive odd/even bits.
3) NEIGHB, LAC, LEND, E123: The routing temp tables, and line up/down flags.
4) RUT: Route Use Table, =0 for us, <0 IMP dead, otherwise branch to TASK Store and Forward.

5) NONE, RSEX, TSKFLG: communication registers for TASK and input routines.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.6.1

NAME

TASK Store and Forward

FUNCTION

1) Select output line for the packet.
2) Search for an output slot on that line.
3) Check and update the storage utilization counts.
4) Assign the packet to an output slot.
4a) Trace the packet if necessary. Acquire a trace block from the free trace list. Put a pointer to the packet in the block. Copy the packet header and input time and trace time into the block.
5) Put the packet on an output queue.
6) Call IMP to Modem if it is idle.

CONTROL STRUCTURE

TASK Store and Forward is designed to run in the minimum time necessary to perform functions 1-6. The routine is almost completely straight line code. Function 2 has the from of a loop. Function 4a is performed by the subroutine TSUB which is also called by TASK Reassembly. Functions 3, 5, and 6 run with interrupts locked, otherwise interrupts are enabled.

ENTRY POINTS

Function 1 is actually a test to determine whether the packet is destined for this IMP , in which case the code branches to FORUS, or whether it is traffic for some other IMP.

EXTERNAL CALLS

6) Software interrupt of IMP to Modem if it is idle.

INITIALIZATION

None.

CLEANUP

The modem output slots and queues are garbage-collected in DEDL when a line goes down (see 5.2).

DATA STRUCTURES

LOCAL DATA

1) OURP, OURL: Our route to send the packet out.
2) I2MSLT: Output slot to use for the packet.
4) I2MSLP, I2MBIT: Channel corresponding to slot, and bit pointer for associated odd/even bit.

SHARED DATA

1) RUT: Route Use Table.
2) I2MTAB, I2MEND: Output slot pointers.
CHFREE: Bits indicate slots in use.

3) NSFA, NSFS, MAXS: Store and Forward storage utilization counts and limit.
   NFA, NFS, NALA, NALS, MINF: Free storage utilization counts and limit.
4) TSEX, LEND: Transmit odd/even bits and line high end/low end flag.
4a) TRCTBL: Free and active trace block table.
5) EMPQ, EMQ: End of modem priority and regular queues.
6) NONE: Modem output busy flag.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.6.2

NAME

TASK For Us

FUNCTION

1) Set up local variables, compute message block, pointer, perform block check and set or flag if it fails. Range check the message number and set a flag if it fails.
2) Branch out on raw packets and network control (incomplete query, out of range, and message block protocol) messages. Discard data messages that failed the range check or block check.

For transmissions:

3) Check for duplicates by inspecting the current state of the reply table, and discard duplicates.
4) Dispatch on destination Host up or down, single or multi- packet, and request for allocation or not.
5) If the Host is down, reclaim any storage allocated for the message and return a destination dead message.
6) For single packet requests, process them as messages if there is enough storage, otherwise mark the reply table as request received.
7) For single packet messages, find the reassembly blocks in the reassembly table and add the packet to it if it is not a duplicate. If the message is complete, check the message number to see if it is the next to go into the Host. If it is, put the message on the Host queue (see 10, below).
8) For multi-packet requests, mark the reply table as request received and go on to the next message loop (11 below).
9) For packets of a multi-packet message, find the reassembly block in the reassembly table and add the packet to it if it is not a duplicate. If the message is complete, treat as a single packet message.
10) When putting a message on the Host queue, remove the packet(s) from the reassembly block and free the reassembly block. After the packet(s) are placed on the Host queue, call Host output, increase the receive message number, and go to the next message loop.
11) In the next message loop, check the reply table state for the next message. If the state is idle or calling for Background action, quit. Otherwise, if the state is request received or abnormal message received, mark as reply for Background action, increment the message number, and loop. Otherwise, search the reassembly table for the message and give it to the Host, or quit if none is found.

For raw packets:

12) If there is no room, discard the packet. Otherwise give directly to Host and then quit.

For incomplete queries:

13) If the message number of the incomplete transmission is the next number to go in to the Host, send back an incomplete reply message. Otherwise, if the message number is in the previous window of 8 messages, send a duplicate reply if in idle state, and an out-of-range if in any other state. Otherwise send an out-of-range.

For replies:

14) Check for duplicates by inspecting the outstanding message bits, and discard duplicates.
15) Find the Transaction Block Table (TSB) entry.
16) If a single packet allocate, mark the TSB entry, remove the packet pointer, and put it on the reply queue for retransmission by background.
17) If a multi-packet allocate, increment the block and Host allocate counts and reset the Host allocate timer (if it is not already running).
18) If not a single packet allocate or reply to a multi-packet request, put the Transaction Block on the Host control message queue.
19) If not a single packet allocate, clear the TSB entry, mark the message number completed, and call the Host input routine if the Host-to-IMP process is waiting for some resource.

For out-of-range messages:

20) Discard the message if either the block or check failed, or if it is a duplicate. Mark the transmit message block so that a connection reset can be initiated by Background.

For block protocol messages:

21) If a get-a-block, perform Host access check and Host status check, and send a got-no-block message if either fail. Otherwise search for a receive message block and send a got-a-block if one is found.
22) If a got-no-block, discard if block check failed, otherwise notify the Host and mark transmit message block for deletion by Background.
23) If a got-a-block, discard if block check failed, otherwise enable transmit message block for use in sending messages.
24) If a reset request, ignore it if any messages are outstanding. If the block check failed, send a reset. Otherwise mark the transmit message block so that a connection reset can be initiated by Background.
25) If a reset, then if the block check succeeded, garbage collect all receive resources, including the message block. Always send a reset reply.
26) If a reset reply, garbage collect all transmit resources, including the message block.

CONTROL STRUCTURE

TASK For Us is a tree-structured group of code, most of which is straight line with some subroutine calls. Many sections run with interrupts locked, some because of shared data, others because of shared code.

ENTRY POINTS

In TASK, when the destination of the packet is determined to be this IMP, the code branches to TASK For Us.

EXTERNAL CALLS

1) HOSTNO: Get local Host number; also called by Timeout.
3) BLKAGE: Set message block age back to 4; also called by Host-to-IMP and Background.
5,13) REASF: Free up reassembly block and packets; also called by Timeout.
5-9) RALLYP: Put an entry into the reply table; also called by IMP-to-Host and Background.
10) TSUB: Trace packet.
15) TSBGET: Search the TSB Table; also called by Background and Host-to-IMP.
10,19) IHSB: Call IMP-to-Host.
19) HISB: Call Host-to-IMP.
21) HSTATR: Get Host up/down attributes; also called by Background.
BLKSRC: Search for message block; also called by Host-to-IMP.
GETLUS: Get local block/use number; also called by Background.
25) RESETR: Garbage collect receive resources; also called by Timeout.
26) RESETT: Garbage collect transmit resources: also called by Timeout.

INITIALIZATION

4) Hosts are initialized to be down.

CLEANUP

Blocks and message numbers are timed out. Reset requests, resets, and incomplete queries are transmitted and retransmitted by Background.

DATA STRUCTURES

LOCAL DATA

THIS: Packet pointer
THISB: Block pointer.
MESTB1: Message number pointer.
MESTB2: Message bits pointer.
SEQNUM: Sequence number for this packet.
MESBIT: Bit corresponding to this packet.
MESSID: Message type.
ORB: Our reassembly block.

TSKTMP: Our packet number.
TEND: Temporary queue end pointer.
READY: Pointer to next packet to give to host.
READYE: Queue end pointer corresponding to READY.
T2HCNT: Packet counter for reassembly.
T2HS12: Message size (in bits) for reassembly.
SOURCE: IMP number of source of this packet.
TSBPTR: Pointer to TSB table for this reply.
LOCHST: Number of host to give message.
RALGT1: Reply type.
BLKFLG: Block check flag.
RNGFLG: Range check flag.
TSKHC1, TSKHC2: Foreign HACMEM and HACCOM.
BLKRST: Reset routine pointer.

SHARED DATA

4) HIHD: Host up/down indicator.
5-11,13,19) NREA, NRES, NALA, NALS: Storage utilization counters.
10) EHQ, EHPQ: Host regular and priority queues.
11) BOFLAG: Reply table active flag.
13,16,21,24,25) ERPQ: Task reply queue for retransmission by background.
17) HIALLT: Host allocate counter and timer.
18) EHRQ: Host control message queue.
21) BLKSLC: Software lock on BLKSRC routine.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7

NAME

Background

FUNCTION

1) Call each Fake Host input process.
2) Call each Fake Host output process.
3) Call each Back Host process.
4) Verify that TIMEOUT is running, initiate a reload if not.
5) Run the nice-stop code if necessary.
6) Call TIP Background (TIPs only).
7) Calculate the light register display.
8) Call VDH Background (VDH IMPs only).
9) Call Satellite Background (Satellite IMPs only).

CONTROL STRUCTURE

Background runs in a loop, performing all of its functions repeatedly. All interrupts are enabled for the most part, so Background runs when no other more important processes are running. Thus its functions can be characterized as periodic but not critical. Functions 1, 2, 3, and 5 are called as coroutines; each background call returns where the previous call left off.

ENTRY POINTS

Entered from Initialization and run continuously thereafter.

EXTERNAL CALLS

1) Return by calling DOZE.
2) Return by calling WAIT.
3) Return by calling SLEEP.

INITIALIZATION

The coroutine entries for 1, 2, 3, and 5 are initialized.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

1,2) FAKENO: The number of the Fake Host last run.
3) BACKNO: The number of the Back Host last run.
1) DZTB: Table of saved return addresses.
2) WTTB: Table of saved return addresses.
3) SLTB: Table of saved return addresses.
4) WDTOLD, WDTBAK: Saved TIME, countdown on BACK cycles with TIME unchanged.

SHARED DATA
None.

I/O PERFORMED
7) Light display output from the A register.

PROGRAM DESCRIPTION 4.7.1

NAME

SUCK

FUNCTION

Simulate the IMP-to-Host interface hardware for the Fake Hosts. Wait until the next output is sent, then fetch each word through the output pointer, and increment the pointer. If the buffer is empty (the output and end pointers are equal) and it is a final output, set the end-of-message indicator for the Host. If the output and end pointers cross, give an output interrupt. Return when a new word is ready for output.

CONTROL STRUCTURE

SUCK is called by each Fake Host output process as a subroutine to receive one word from the IMP. It returns when a word is ready, and makes a coroutine return to the main background loop if there is no output ready.

ENTRY POINTS

Called by:
Fake Host 0: TTY output.
Fake Host 1: DDT output.
Fake Host 2: PARAMETER CHANGE and PACKET CORE output.
Fake Host 3: DISCARD.

EXTERNAL CALLS

IHSB: Software interrupt of IMP-to-Host.
WAIT: Coroutine return to Background.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

SUCT: Table of saved return addresses.

SHARED DATA

IHBB: Simulated DMC output pointers.
IHBC: Simulated DMC output end pointers.

I/O PERFORMED

None. (Simulated I/O includes output transfers, and software interrupt on output buffer empty.)

PROGRAM DESCRIPTION 4.7.1.1

NAME

TTY SUCK

FUNCTION

Outputs messages sent to fake host TTY
1) as ASCII characters if parity bit is on, or
2) as octal numbers, after an optional
3) octal print of the leader, utilizing a
4) common send character routine.

CONTROL STRUCTURE

A strung out loop which first locks out TTY input, saves the source if foreign, and commits the octal print bit (2) to memory before printing anything. It then octal prints the header (3) if TTY JAM last sent a multi-character message and the host simulator flag was on then. Otherwise it skips over the leader noticing if the link word was the last word and if it is, checks the subcode and types a backslash if it is non-zero. If the link is not the last word, the non last data words are either printed out as octal numbers (2) using an octal print routine or as ASCII characters (1) using a routine (4) shared by the octal print routine. This common send routine (4) rejects characters with zero parity and sucks up the rest of the message if output is in progress. This requires that other TTY sucks keep the send routine informed as to whether they are the last so send doesn't mistakenly throw away the next message!

ENTRY POINTS

Coroutined with SUCK/WAIT.

EXTERNAL CALLS

None.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

TTOW: Last word returned by SUCK (1,2,3,4).
TTNM: Set while processing last word (1,2,3,4).
OCTL: Set minus for octal print (2).
OC01: Octal print digit counter (2,3).
OC03: Octal print temporary (2,3).

SHARED DATA

HSGO: HSFG at beginning of last TTY JAM message.

WHOTTY: Last foreign source.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7.1.2

NAME

DDT SUCK

FUNCTION

Feed messages a character at a time to DDT.

CONTROL STRUCTURE

DDT SUCK is coroutined with DDT JAM to form one DDT process. The DDT SUCK process saves the SUCK message leader for DDT JAM and resets BBNF if the message is from TENEX, or TTY at IMP 40 or 30. It then breaks each word up into characters and calls a subroutine which gives them to DDT JAM (via DDTC) and waits for them to be taken (DDT JAM zeros DDTC when it takes a character.) However if the parity bit is not set it returns immediately, and if it is a break it sets the DDT JAM wait return to the DDT reset address and resets the suppress output flag (DDTI which TTY JAM may have set.) At the end of a message it sets a flag at which DDT JAM will look before its next read and if it is set will close its JAM message.

ENTRY POINTS

Coroutined with SUCK/WAIT.

EXTERNAL CALLS

None.

INITIALIZATION

Done by TTY JAM.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

DINW: Word returned by SUCK.

SHARED DATA

DINC: Character for DTT input routine.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7.1.3

NAME

Parameter Change and Packet Core output

FUNCTION

1) Accept new values for the parameters governing the operation of the Trace and Statistics programs.
2) Accepts packets which control the loading and dumping of core areas within the IMP, or which are converted to block format and sent to a specified (malfunctioning) neighbor.

If the first word of the message is positive, function 1 is implied, and gives the number of the parameter to change. The second word then gives the new value for the parameter. Otherwise, function 2 is implied, and the first word determines whether the message is to be formatted as a PACKET CORE message and sent out the appropriate modem, or processed as a SETUP or CORE message.

CONTROL STRUCTURE

Function 1 has the form of a loop, with a coroutine call to SUCK functioning as the implied wait. Interrupts are enabled.
Function 2 starts out similarly, but if a PACKET CORE is to be sent, additional coroutine calls to WAIT occur until the modem flag becomes free.

ENTRY POINTS

Coroutined with SUCK/WAIT.

EXTERNAL CALLS

2) GETFRE: Get free buffer routine.

INITIALIZATION

1) All parameters are initialized to zero. Periodically, those parameters necessary for NCC reports and diagnostic messages are set to their nominal values.
2) The flag which indicates whether a new message must wait for the output of a previous PACKET CORE is cleared.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

1) BESTFL: A pointer into PARAMT.
2) PKCBFR: Modem out buffer pointer.
   PKCBFF: Modem out start pointer, (0=modem free).
   PKCMDN: Modem number (0 - 4).

PKCLDR: Source host.
PKCLD1: Source IMP.
PKCLIN: Saved local line no. (1 - 5).
PKCTMP: Temporary pointer.
PKCTMC: Temporary counter.

SHARED DATA

1) PARAMT: The table of parameters:

0- TRON: Trace on/off
1- SNON: Snapshot statistics on/off
2- SON: Cumulative statistics on/off
3- MGON: Message Generator on/off
4- DIAGON: diagnostic reports on/off
5- TPON: NCC reports on/off
6- PTON: Packet Trace on/off
7- TLINK: Trace link
10- STATL5: Links for: Snapshot statistics
11- Cumulative statistics
12- Message Generator
13- Diagnostic reports
14- NCC reports
15- TDSTI: Trace destination IMP
16- STATL4: Destination IMPs for: Snapshot statistics
17- Cumulative statistics
20- Message generator
21- Diagnostic reports
22- NCC reports
23- TDSTH: Trace destination handling type/host
24- STATL3: Destination handling type/hosts for: Snapshot statistics
25- Cumulative statistics
26- Message generator
27- Diagnostic reports
30- NCC reports
31- TDSTF: Trace destination flags/message type
32- STATL2: Destination flags/message types for: Snapshot statistics
33- Cumulative statistics
34- Message generator
35- Diagnostic reports
36- NCC reports
37- TF: Auto Trace frequency
40- STATF: Frequencies for: Snapshot statistics
41- Cumulative statistics
42- Message Generator
43- Diagnostic reports
44- NCC reports
45- MGNL: Message Generator length
46- PTF: Packet trace frequency
47- RTTUNT: Round Trip time units
50- ATDSTI: Auto Trace destination IMP
51- ATDSTH: Auto Trace destination host
52- ATSRCI: Auto Trace source IMP

53- ATSRCH: Auto Trace source host
2) PKCFRH: Foreign Host.
PKCFRI: Foreign IMP.
PKCFRL: Foreign message ID.
PKCFRM: Foreign line number.
PKCADR: Current core address.
PKCSIZ: Current number of core words left.
PKCSTF: Send SETUP flag (>0=>send it)
PKCSRF: Send/receive flag (<0=>receiving timeout, =0=>idle, >0=>sending frequency).
PKCNBR: Dead neighbor table.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7.1.4

NAME

Discard

FUNCTION

Discard is a process which simply accepts messages from the IMP via SUCK. It is used to guarantee the return of a RFNM or Incomplete Transmission to a source Host, by virtue of the standard IMP to Host mechanisms and the fact that Discard is always a responsive Host. In particular, when a destination Host takes too long to accept a message, or goes down when the IMP is holding messages for it, the messages are marked incomplete and put on Discard's queue. When the messages are accepted by Discard and thrown away, an Incomplete Transmission is automatically sent back to the source Host. In addition, Discard looks for arriving RFNMs and resets the Software Watchdog timer mechanism if one arrives.

CONTROL STRUCTURE

Discard has the form of a loop, with a coroutine call to SUCK functioning as the implied wait. Interrupts are enabled.

ENTRY POINTS

Discard runs whenever SUCK has a word from the IMP.

EXTERNAL CALLS

None.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

None.

SHARED DATA

WDTIME, WDSTAT: Timeout and state counter for Software Watchdog timer mechanism.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7.2

NAME

JAM

FUNCTION

Simulate the Host-to-IMP interface hardware for the Fake Hosts. Receive a word from the Host, store it through the input pointer, and increment the pointer. If the end-of-message indicator is on, or the buffer is full (the input and end pointers cross), give an input interrupt. Wait until a new input is possible, and then return.

CONTROL STRUCTURE

JAM is called by each Fake Host input process as a subroutine to send one word to the IMP. It returns when that word has been taken and another input is logically possible. If another input is not possible, it makes a coroutine return to the main background loop.

ENTRY POINTS

Called by:
Fake Host 0: TTY input.
Fake Host 1: DDT input.
Fake Host 2: Trace and Packet Core input.
Fake Host 3: Statistics input.

EXTERNAL CALLS

HISB: Software interrupt of Host-to-IMP.
DOZE: Coroutine return to Background.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

GAMT: Table of saved return addresses.

SHARED DATA

HIBB: Simulated DMC input pointers.
HIBC: Simulated DMC input end pointers.

I/O PERFORMED

None. (Simulated I/O includes input transfers, and software interrupt on end-of-message and input buffer full.)

PROGRAM DESCRIPTION 4.7.2.1

NAME

TTY JAM

FUNCTION

1) Process teletype interrupts and input typed characters or echo backslash if last character not taken.
2) Send single character message to crosspatch destination, and if a break, reset crosspatch destination to local DDT.
3) Send multi-character messages to message destination.
4) Send octal numbers within multi-character messages.

CONTROL STRUCTURE

Function 1 is performed by an interrupt routine which dismisses output completion interrupts for tty output and backslashes, types a backslash if the last character has not been taken, and reads the TTY and leaves the character, and an indication of its arrival for the TTY fake host, i.e. the TTY JAM background process. Function 2 is performed by two coroutines one of which either returns with a character with its parity bit ored on or dozes while waiting for the teletype interrupt routine. The other checks if the character is the message escape character ";" and jumps into a function 3 coroutine if it is. Otherwise it JAMS the crosspatch leader, the character left justified, resetting the crosspatch header to local DDT for a break, and padding. Function 3 is performed by two coroutines one of which uses function 2's read/doze routine to get characters, jumps back to the function 2 jam coroutine for a ";" and jumps into the function 4 routine for the number escape character ":". The other coroutine JAMS the message leader, and then loops, one character to a word. Function 4 is performed by the function 3 read coroutine and a coroutine which accumulates an octal number from the three least significant bits of successive characters, terminating on carriage return, echoing a linefeed, and jumping back into the function 3 JAM coroutine.

ENTRY POINTS

Coroutined with JAM/DOZE.

EXTERNAL CALLS

None.

INITIALIZATION

Zeros OTGO, HSFG, HSGO, and TTCH; and sets TTY crosspatch destination to local DDT. Zero DINC and set DDT destination to local TTY for DDT. Set interrupt mask enabling TTY and disabling all others.

CLEANUP
None.

DATA STRUCTURES
LOCAL DATA
TTFG: Interrupt flag word.
TINA: Interrupt A register save.
TTCR: Interrupt character.
TTCH: Read/DOZE character.
TTIW: Word so far.

SHARED DATA
OTGO: Output in progress flag.
HSFG: Host Simulator flag.
HSGO: Host Simulator flag at beginning of last message.

I/O PERFORMED
TTY input and output.

PROGRAM DESCRIPTION 4.7.2.2

NAME

DDT JAM

FUNCTION

Output DDT characters to the last DDT SUCK source.

CONTROL STRUCTURE

DDT and its input and output routines are run on the DDT JAM process. The DDT input routine first tests if it has encountered a message end and if it has clears the message end flag and calls part of the DDT output routine to send off any output that has been done (i.e. end the current JAM message.) If the test fails or once the output routine returns, the input routine checks if DDT SUCK has left it a character (in DDTC.) If it has it returns it to DDT (having zeroed DDTC.) DDT and the DDT output routines are coroutines. The output routine is transparent to the A, B, and X registers. The saving/restoring is common to all of the output routine's entries/returns. The output routine JAMs the header DDT SUCK has saved and then returns/calls DDT for characters which it JAMs (oring on parity) one at a time. It breaks output up into many messages for some commands and for some long-winded input messages (recall that the DDT input routine closes the DDT JAM message on encountering an input message end.) The DDT output routine performs an additional function: if a flag which TTY SUCK sets is set, it resets DDT by altering the output routine's return address and returning (which allows the TTY to interrupt the local DDT.)

ENTRY POINTS

Coroutined with JAM/DOZE.

EXTERNAL CALLS

None.

INITIALIZATION

Done by TTY JAM.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

DOTW: Output word save.
DOTA: Saved A register.
DOTB: Saved B register.
DOTX: Saved X register.
DCNT: Output message word count.

SHARED DATA

DINC: Input character.

DEND: End of input message flag.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7.2.3

NAME

Trace and Packet Core input

FUNCTION

1) Initiate a trace message to the selected destination.
2) Find complete trace blocks on the active trace queue.
3) Copy them into the message.
4) Return the blocks to the free trace queue.
5) Copy a PACKET CORE from its queue into a message.
6) Get a core segment and copy it into a message.

CONTROL STRUCTURE

If there is anything on the active trace queue, function 1 is performed, followed by functions 2, 3, and 4 in a loop until the end of the active trace queue is reached. The message is then closed off. Functions 2 and 4 run with interrupts locked. If there is anything on the Packet Core queue, the first message is removed from the queue and function 5 is performed. Function 6 is performed by examining the Packet Core variables and, if appropriate, sending a single (to dead IMPs) or multi-packet (to live IMPs) message containing a segment of the core image. The program goes to sleep for one background loop after function 6 and then starts again with function 1.

ENTRY POINTS

Called as a background coroutine once every background loop.

EXTERNAL CALLS

PKCCLC: Reset Packet Core process to idle.

INITIALIZATION

PKCCLC is called to clear the Packet Core state variables.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

3) T2BX: Copy loop counter.
   T3BX: Counter for blocks copied.
2) OLD2: Queue pointer used in search.
2-4) OLD1: Packet pointer used in copy.
5) PKCQBF: Queue buffer pointer.
   PKCPTR: Temporary pointer.
   PKCCNT: Temporary counter.
   PKCLK1: 100 microsecond clock saved value.
   PKCLK2: 102.4 millisecond clock.

SHARED DATA

1-4) TTF, STRQ: Free and active trace blocks.
TTO: Trace overflow counter.

5) SPCQ, EPCQ: Packet Core queue.
Also, see section 4.7.1.3

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7.2.4

NAME

Statistics

FUNCTION

1) Detect transitions in the cumulative statistics on/off indicator and update the statistics-gathering locations. Turn on TRBL and DIAG and set their parameters.
2) Check each active statistics program to see if it is time to call it. If it is, send out the message leader.
3) Call SNAP, the Snapshot Statistics program, if necessary.
4) Call SEST, the Cumulative Statistics program, if necessary.
5) Call GENM, the Message Generator program, if necessary.
6) Call TRBL, the NCC Trouble Report program, if necessary.
7) Call DIAG, the NCC Diagnostic Report program, if necessary.

CONTROL STRUCTURE

The five Statistics programs are multiplexed on a single fake host port. The Statistics slot is a coroutine in the background loop which performs function 1 and then performs the function 2 for each of the 5 programs, using shared code. Both the functions run with interrupts enabled. When it is time to run a given statistics program, then functions 3, 4, 5, 6, and 7 may be performed. These programs run with interrupts enabled for the most part.

ENTRY POINTS

Called as a background coroutine once every background loop.

EXTERNAL CALLS

None.

INITIALIZATION

1) The saved copy of the Cumulative Statistics on/off flag is initialized to be off, and the statistics-gathering locations are initialized to their nominal contents.
2) All the statistics programs are initialized to be off.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

1) SOFO: Saved copy of the Cumulative Statistics on/off flag.
   SB1. SC1, SW1: Tables for statistics-gathering locations, their nominal contents, and their contents when cumulative statistics are turned on.
2) OLDS: Table of the times that each statistics program was last run.
3) SNPC, SNPP: Host queue counter and pointer.

SHARED DATA

1) SON: Cumulative Statistics on/off flag.
2) SNON, SON, MGON, TPON, DIAGON: The on/off flags.
   STATF, STATD, STATL: Tabled parameters for each statistics program, giving frequency, destination, and link.
3) TIME: Local time.
   SHQ: Host queues.
   NFA, NFS, NSFA, NSFS, NREA, NRES, NALA, NALS: Storage utilization counters.
   RUT, RST: Route Use Table and Route Send Table.
   SATSNP: Table of Satellite IMP snapshot locations.
4) SYNC: Global time.
   STTB: Table of statistics counters.
   SATCUM: Table of Satellite IMP statistics counters.
6) HIHD: Host alive/dead flags for each host.
   SWS: Switch setting word.
   RSFNCC: Restart/reload indicator.
   HLTLOC, HLTA, HLTX: Saved PC, A, X from last halt.
   NFA, NFS, NSFA, NSFS, NREA, NRES, NALA, NALS.
   VERS: IMP program version number.
   HOST34: IMP configuration word.
   TIPVER: TIP program version number.
   HLNM, HLSNT, HLRCVD: Number of host being tested, number of test messages sent and received.
   LINE, NEIGHB: Tables for each line, giving up/down status and neighbor imp number.
   RTSSNT, E123, E321: Tables of number of routing messages sent, received, and missed for each line.
   THRUPT: Table of number of acks per line.
   NTRTAB, HTPTBL: Table of pointers and a table of host throughput counters.
   DIAGQ: queue of error packets waiting to be sent to the NCC.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.7.3

NAME

Back Hosts

FUNCTION

The Back Hosts serve as the source of several important control messages.

1) BACK0: Send RFNMs, allocates, RFNM/allocates, destination deads, and incomplete replies.
2) BACK1: Monitor transmit message blocks and mark blocks for reset.
3) BACK2: Monitor receive message blocks and send off reset requests.
5) BACK4: Send messages from reroute and reply queues (rerouted messages or Task replies).
6) BACK5: Send incomplete transmissions, givebacks, resets, and get-a-block messages.

CONTROL STRUCTURE

The Back Hosts are a series of coroutines called from the main Background loop. Each routine determines if a particular kind of message needs to be sent. If one does, it formats the appropriate message, gives it to TASK and waits for it to be accepted. If at any time processing cannot continue, the routine makes a coroutine return to the Background.

ENTRY POINTS

Coroutine entrance from Background.

EXTERNAL CALLS

GIVTSK: Call the TASK routine with a packet, generating its checksum, and wait for it to be accepted.
SLEEP: Return to the Background loop.
GETFRE: Get a free buffer.
1) RALLYP: Put an entry into the reply table.
HSTATR: Get Host status.
3,6) GETLUS: Get local block/use number.
3) BLKAGE: Reset block age to 4.
5) GETQ: Get a packet off a queue.
6) GETALL: Get an allocate from the Host.
MESGET: Get a message number.
TSBGET: Get a TSB table entry.

INITIALIZATION

The coroutine entries, tabled in SLTB, are initialized for each routine.

CLEANUP

1) No RFNM is delayed more than 1/2 second by waiting for a piggy-backed allocate.

DATA STRUCTURES

LOCAL DATA

1) BACKOP: Receive message block pointer.
   BOMESS: Saved message number.
   BOPOSN: Reply position counter.
   BOTYPE: Saved RTYPE.
   BOSTAT: Saved RSTATE.
   BOPASS: Full pass counter.
2) BACK1P: Transmit message block pointer.
   B1TIMR: Sleep timer.
   B1CUTC: Cutoff counter.
   B1CYCC: Cycle counter.
   B1CAGC: Age counter.
3) BACK2P: Receive message block pointer.
   B2TIMR: Sleep timer.
   B2CUTC: Cutoff counter.
   B1CYCC+1: Cycle counter.
   B1CAGC+1: Age counter.
   B2TMP1: SEQH word for reset request.
   B2TMP2: MIDH word for reset request.
6) BACK5P: Transmit message block pointer.
   BACK5I: Immediate action pointer.
   B5BLKP: Temp block pointer.
   B5TYPH, B5TEMP, B5SRCH, B5SEQH, B5PKTH, B5DSTH, B5MIDH, B5HMEM, B5HCOM: Message header.

SHARED DATA

TSKFLG: A communication flag for each Back Host which indicates whether TASK accepted or rejected the last input.

1) BOFLAG: Reply active flag.
   NREA, NRES, NALA, NALS: Storage accounting timers.
   REASTB: Reassembly block table.
2) B1FCNT: Free transmit message block count.
3) B2FCNT: Free receive message block count.
   B2NCNT: Additional needed receive message block count.
5) SRPQ, ERPQ, SRRQ, ERRQ: Reply and reroute queue.
6) HACMEM, HACCOM: Host access control words.
   HIALLT: Host allocate timer.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.8

NAME

Very Distant Host (VDH)

FUNCTION

To implement the Very Distant Host interface described in Appendix F of BBN Report 1822. This includes handling the VDH modem interfaces, implementing the Reliable Transmission Package described in this Appendix, and efficiently interfacing the Reliable Transmission Package to IMP to Host and Host to IMP while at the same time simulating the Regular Host Interface as far as Host to IMP and IMP to Host are concerned. The VDH routine can be used with any Host and modem combination, precisely which being determined by the contents of the hardware configuration cards.

PROGRAM DESCRIPTION 4.8.1

NAME

VDH Initialization

FUNCTION

1) To initialize the entire VDH package when the system is restarted.
2) To initialize a VDH host when a background flag indicates that this must be done.
3) To reinitialize the VDH acknowledgement system when a VDH phone line has failed or when IMP to Host flaps a ready-line.

CONTROL STRUCTURE

Functions 1, 2, and 3 are non-reentrant subroutines.

ENTRY POINTS

1) The entry point is VDH.I, called by IMP background through "VDH..I".
2) VDH background calls VD.I. when background flags indicate that a new host might need to be brought up.
3) The entry point is VD.REI, called by function 2, and by VDH Timeout when the VDH phone line goes dead, IMP to Host flaps the ready-line, or a "spurious ack" is detected.

EXTERNAL CALLS

1) VD.I calls FLUSH to return available space between VDH pages to the free buffer pool.
2) VD.I. calls VD.REI to have the acknowledgement system reinitialized.
   VD.REI calls FLUSH to return unacknowledged buffers.

INITIALIZATION

1) The call to VD.I is placed into IMP background when the VDH package is loaded.

2-3) None.

CLEANUP

None.

DATA STRUCTURES

None specific to VDH Initialization.

LOCAL DATA

VDIHN: Host being initialized.
VDIMN: Corresponding modem number.
VD.IT: Temp storage.

SHARED DATA

All tables or variables initialized by VDH Initialization are, of course, in some sense shared

with the other VDH routines. Therefore, appropriate interlocks must be taken with calls to VD.REI.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.8.2

NAME

VDH Input Interrupt

FUNCTION

1) To process modem input interrupts from a VDH phone line and to initiate new modem inputs for this same line.
2) To process acknowledgements.
3) To note the arrival of HELLO's and I-HEARD-YOU's.
4) To pass packets received from a phone line to VDH Background.
5) To mark for acknowledgements to be sent.
6) To detect duplicate packets received.

CONTROL STRUCTURE

VDH Input Interrupt is a non-reentrant subroutine called by a VDH modem input interrupt. Included is one local subroutine which is called to process acknowledgements.

ENTRY POINTS

The entry points to VDH Input Interrupt are VD.IIn (where n runs from 0 through 4) which are called by a VDH modem input interrupt. VDH Input Interrupt runs on the same priority as the other modem interrupts and runs locked. VD.AP is the entry point to the local subroutine used to process acknowledgements.

EXTERNAL CALLS

If a spurious ack is received, HLTNCC is called to send a trap to the NCC and VD.REI is called to resync the acknowledge sequence.

INITIALIZATION

All initialization, including setting up of interrupt pointers, is performed by VDH initialization.

CLEANUP

None other than that already mentioned for spurious acks.

DATA STRUCTURES

LOCAL DATA

The keys and registers are saved in VD.IK, VD.IA, and VD.IX. VD.IIB, VD.RBL, VD.CWP, and VD.RCN are used to save the input buffer pointer, the buffer length, the input buffer control word, and the receive channel number. VD.IT1 and VD.IT2 are temporaries. VDIIHN and VDIIMN keep the host and modem number being processed.

<u>SHARED DATA</u>

VDH Input Interrupt accesses the free queue and NFS, accesses the counters VD.R and VD.T which are shared with other VDH routines, and accesses the tables VD.TOE, VD.TB, VD.RE, VD.ROE, and VD.RB which are shared with the other VDH routines. VDH.IHY and VDH.HLO, the received I-HEARD-YOU and HELLO flags, are shared with the Output Interrupt and Timeout routine, respectively.

<u>I/O PERFORMED</u>

Does VDH modem inputs.

PROGRAM DESCRIPTION 4.8.3

NAME

VDH Output Interrupt

FUNCTION

1) To process modem output interrupts from the VDH phone line and to initiate modem outputs for this line.
2) To build acks.
3) To retransmit unacknowledged packets.
4) To free buffers of acknowledged packets.
5) To pass packets from VDH Background to the phone line.
6) To send HELLO's and I-HEARD-YOU's as necessary.

CONTROL STRUCTURE

VDH Output Interrupt is a non-reentrant subroutine called by a VDH modem output interrupt and by VDH Background, the latter when an output interrupt sequence must be restarted. Included are two local subroutines: one to build the acknowledge field in the control word and the other to service each output channel in turn.

ENTRY POINTS

The entry points to VDH Output Interrupt are VD.OIn (where n runs from 0 through 4) which are called by the VDH modem output interrupt or by function 4 of VDH BACKGROUND. VDH Output Interrupt has the same priority as the other modem output interrupts. VD.OIT is the entry point to the subroutine used to build the acknowledge field in the control word. VD.OIS is the entry point to the subroutine used to service each output channel in turn. The subroutine VD.OIS is peculiar in that when there is nothing to do for a particular channel, the subroutine returns, but when there is something to do for a given channel, then rather than returning, the subroutine jumps to VD.OI3.

EXTERNAL CALLS

VD.OIS calls FLUSH to return acknowledged buffers.

INITIALIZATION

Initialization, including setting up the interrupt pointers, is performed by VDH initialization.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

VD.OB and VD.CW are used to save pointers to the output buffer and to build the control word field. VDOIHN and VDOIMN are the host and modem being processed.

SHARED DATA

VDH Output Interrupt accesses the free queue via FLUSH, accesses the counters VD.R and VD.D shared with the other VDH routines, and accesses the tables (e.g., VD.TE and VD.TB) shared with the other VDH routines. VD.HLO and VD.SH, the send I-HEARD-YOU and HELLO flags, are shared with the Input Interrupt and Timeout routines, respectively.

I/O PERFORMED

Does VDH modem outputs.

PROGRAM DESCRIPTION 4.8.4

NAME

VDH Timeout

FUNCTION

To decrement VD.R and VD.D as appropriate, to test an IMP/Host ready-line, and to call VD.REI if an IMP/Host ready-line has been flapped or if a VDH phone line is detected as dead.

CONTROL STRUCTURE

VDH Timeout is a reentrant subroutine. VD.SBR is the table of coroutine dispatches for each host.

ENTRY POINTS

The subroutine entry point is VD.TO which is called through VDH3. by IMP Timeout.

EXTERNAL CALLS

VD.REI is called when a VDH phone line goes dead or an IMP/Host ready-line is flapped.

INITIALIZATION

The call from IMP Timeout is initialized by VDH initialization, as are the coroutine entries.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

VDTOHN: Host being processed.
VD.TOC: Counter for outer loop (by host).
VD.TCK: Timers (when to run Timeout for each host).
VD.DWN: Internal ready-line-flap flags.

SHARED DATA

The counters VD.R, VD.SH, VD.IHY, and VD.D are shared with the other VDH routines. The flags VD.RDY are shared with IMP to Host.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.8.5

NAME

VDH Background

FUNCTION

1) To initialize new VDH host(s) at the request of background flags.
2) To pass packets from VDH Input Interrupt to Host to IMP.
3) To pass packets from IMP to Host to VDH Output Interrupt.
4) To wake up VDH Output Interrupt if it has languished.

CONTROL STRUCTURE

VDH Background is a non-reentrant subroutine called from IMP Background. The subroutine is constructed of three essentially straight line sections of code to handle each of the three functions. An attempt to run all three sections is made each time through VDH Background.

ENTRY POINTS

The entry point to VDH Background is VD.B which is called out of IMP Background via "VDH2.".

EXTERNAL CALLS

1) VD.I. is called when legal bits in VDHRSF are found to be set.
2) FLUSH is called to release the buffer the leader came in and to release the buffer Host to IMP had up for input. An interrupt to Host to IMP is faked via a call through VD.HII when a buffer is ready for Host to IMP.
3) GETFRE is called to get a free buffer into which to copy the buffer IMP to Host has set up for output. After each buffer has been sent, an interrupt to IMP to Host is faked via VD.HOI.
4) If VDH Output Interrupt does not have an interrupt pending, an interrupt is faked via VD.OI.

INITIALIZATION

All performed by VDH initialization.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

The mask is saved in VD.BM -- VDH background runs at IMP to Host level. Function 3 uses VD.HOL to determine whether to send a leader or not. The variables VD.IB, VD.BB, VD.BBT, and VD.BBF are used to swap buffers with the IMP to Host and Host to IMP routines.

SHARED DATA

VDH Background shares all of the various VDH tables with the other VDH routines.

I/O PERFORMED

VDH Background does no actual I/O but simulates the I/O hardware as far as IMP to Host, Host to IMP, and VDH Output Interrupt are concerned.

PROGRAM DESCRIPTION 4.9

NAME

Initialization

FUNCTION

Initialize all data structures, initialize and start up the rest of the IMP system.

CONTROL STRUCTURE

Sequential code entered and executed once only at startup time. The following is a list of initialization procedures in the order performed:

1) Set restart flag properly for trouble reports.
2) Set up MINE.
3) Clear the zero-storage area.
4) Init the zero words, -1 words, and special values.
5) Initialize queue structure.
6) Build free list.
7) Compute reassembly limit.
8) Set background co-routine start addresses.
9) Init the add chain.
10) Init the loadable module connections.
11) Init the Host OCPS and DMC location pointers.
12) Compute IMPMOD (modem ownership), initialize interrupt entrances, and set enabling bits in interrupt masks (MOM, IHM, and HIM) for each modem, based on the hardware configuration cards.
13) Initialize Host status, interrupt entrances, and set enabling bits in interrupt masks (IHM, HIM) for each Host, based on the hardware configuration cards. For TIPs, set the IMP/TIP dependent locations.
14) Initialize routing tables.
15) initialize Timeout.
16) Fire off a trouble report.
17) Start up Modem-to-IMP.
18) Start up IMP-to-Host and Host-to-IMP.
19) Enable all interrupts.
20) Exit to background.

ENTRY POINTS

Entered manually, from the stand-alone after a reload, or from the nice-stop program after the restart flag has been set.

EXTERNAL CALLS

None.

INITIALIZATION

None.

CLEANUP

None.

<u>DATA STRUCTURES</u>

<u>LOCAL DATA</u>

IT1 and IT2 are used for temporary storage.

<u>SHARED DATA</u>

NRSTF: Nice-stop restart flag.
See descriptions of programs being initialized.

<u>I/O PERFORMED</u>

IMP number and real-time clock are read; modem inputs on active channels are performed.

PROGRAM DESCRIPTION 4.10.1

NAME

Power Failure Restart

FUNCTION

To provide automatic recovery in case of a power failure.

CONTROL STRUCTURE

Power failure is detected by an interrupt which cannot be masked off or inhibited. Upon receipt of this interrupt, the system prepares to restart from initialization when hardware detects return of power.

ENTRY POINTS

The hardware interrupt comes to RSTR.

EXTERNAL CALLS

Initialization is begun following return of power.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

TIME, HACSUM, and the index register are usurped to provide for the halt and restart.

SHARED DATA

RSFLAG: Used to signal that power failure occurred.

I/O PERFORMED

The Watchdog Timer is poked.

PROGRAM DESCRIPTION 4.10.2

NAME

Watchdog Timer

FUNCTION

Provide a check on the running IMP system, and if the system should fail to run properly, cause a system reload.

CONTROL STRUCTURE

A properly running IMP sends a status report once each 52 seconds to the NCC. This message is always followed by a dummy message to the local discard (fake Host 3). Thus a RFNM for the dummy message should arrive for fake host 3 at least once each 52 seconds. When this occurs the Watchdog Timer is poked:

1. Hardware. Should two minutes of real time elapse without a poke, the hardware generates an interrupt to the reload code. This interrupt cannot be masked out or inhibited. This feature operates in conjunction with the HALT-INHIBIT switch, which keeps a broken machine running so that the interrupt may occur. (Applies only to DDP-516 IMPS.)

2. Software. The IMP also maintains a software Watchdog Timer counter which is set at the same time as the hardware Watchdog Timer and incremented by slow timeout. Each time a RFNM for fake Host 3 arrives (detected by discard), the timer is reset to 5 min. The software watchdog timer fires if the timer goes to zero. Should this occur, the system jumps directly into the stand-alone code, simulating an interrupt. Background maintains a check to make sure that Timeout is running. If no change is detected in the timeout clock (TIME) in 10000 background loops, it calls the reload code. (Applies to all IMPs.)

ENTRY POINTS

The hardware interrupt and the software "interrupt" come to SWDT.

EXTERNAL CALLS

Initialization is begun following a successful load.

INITIALIZATION

None.

CLEANUP

None.

DATA STRUCTURES

LOCAL DATA

None.

SHARED DATA

RSFLAG: Used to signal that reload has occurred.

I/O PERFORMED

None.

PROGRAM DESCRIPTION 4.10.3

NAME

Stand-alone.

FUNCTION

Provide a small program which operates as a self-contained unit for purposes of dumping or reloading the IMP operating program:

1) Reset all outstanding I/O operations.
2) Request a reload from a neighbor IMP.
3) Send a contiguous area of core memory in PACKET CORE mode.
4) Receive segments of core memory in PACKET CORE mode.

CONTROL STRUCTURE

Functions 1 and 2 are performed in a loop, with function 2 performed several times within each overall loop. The line to the neighbor IMP is either cycled among all possible lines, one line per loop, or held fixed, depending upon initial conditions. Functions 3 and/or 4 can be initiated by an external process, and are then included in the loop. Interrupts are inhibited and I/O events are detected by watching the DMC pointers. by a neighbor IMP. Interrupts are inhibited and I/O events are detected by watching the DMC pointers.

ENTRY POINTS

The stand-alone program can be entered normally after a machine has halted, from the watchdog timer routine, or from the program by the setting of the reload flag (SW3FG). The reload flag can indicate an immediate (panic) reload, in which case the program jumps directly to the stand-alone, or a nice-stop reload, in which case the IMP first gracefully shuts down before entering the stand-alone. The reload flag can be set manually or as a result of a demand reload message from a neighbor IMP.

EXTERNAL CALLS

When the restart (SA.RST) flag is set, the stand-alone jumps to initialization.

INITIALIZATION

The internal variables and PACKET CORE parameters are initialized once each loop.

CLEANUP

None.

AD-A040 675 BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS F/G 9/2
THE INTERFACE MESSAGE PROCESSOR PROGRAM.(U)
MAR 77 DAHC15-69-C-0179
UNCLASSIFIED TECHNICAL INFORMATION-89 NL

2 OF 2

AD
A040675

END

DATE
FILMED
7-77

DATA STRUCTURES

LOCAL DATA

SA.LIN: Our line to reload.
SA.STN: Neighbor IMP number.
SA.FRH: Foreign Host.
SA.FRI: Foreign IMP.
SA.FRL: Foreign link.
SA.FRM: Foreign line.
SA.LCH: Local (neighbor) Host.
SA.LCI: Local (neighbor) IMP.
SA.LCM: Local (neighbor) line.
SA.ADR: Core address for transfer.
SA.SIZ: Transfer size.
SA.STF: Send setup flag.
SA.SRF: Send/receive flag.
SA.ILS: Input buffer location.
SA.INF: Input wait flag.
SA.OCK: 100 microsecond clock saved value.
SA.TIC: 102.4 millisecond clock.
SA.PTR, SA.CNT, SA.TMC: Temp ptr. count, clock event.

SHARED DATA

None.

I/O PERFORMED

1) All I/O channels are flushed by resetting the DMC pointers and then performing a sequence of crosspatch, input, output, and uncrosspatch.
2,4) Output of PACKET CORE setup messages from fixed core area. Input into a fixed, buffer sized area.
3) Output of PACKET CORE core messages from fixed buffer-size core area.

5. Data Formats

A. Old-style Leader Format

Host to IMP

```
 y-{|<<<<<<<<<<<<<<<<<<<|
 |  |____________|XXXXX|  Message-ID (Link)
 |
 |
 `->|_|_|_|_|_._._._|_|_._|_._._._._._._|
      V V V V \_____/ \ / \___________/
      | | | |    V     V        V
      | | | |    |     |        `->Destination IMP
      | | | |    |     `->Destination Host
      | | | |    `->Host-IMP message type
      | | | `->Octal print (for TTY)
      | | `->Trace
      | `->For IMP
      `->Priority
```

IMP to Host

```
  y-{|<<<<<<<<<<<<<<<<<<<|
 y+-{|<<<<<<<<<<<<<<<<<<<|
 ||
 |`->|_|_|_|_|_._._._|_|_._|_._._._._._._|
 |     V V V V \_____/ \ / \___________/
 |     | | | |    V     V        V
 |     | | | |    |     |        `->Source IMP
 |     | | | |    |     `->Source Host
 |     | | | |    `->IMP-Host message type
 |     | | | `->Octal print (for TTY)
 |     | | `->Trace
 |     | `->From IMP
 |     `->Priority
 |
 `-->|_._._._._._._._._._._._._._|_._._._|
      \___________________________/ \__V__/
                   V                   `->Message subtype
                   |
                   `->Message-ID (Link) or Host status
```

B. New-style Leader Format

Host to IMP

```
              ___________________
             |XXXX|    |XXXXXXXX|   New Leader Flag = 7400
         y-{ |<<<<<<<<<<<<<<<<<<|
        y+-{ |<<<<<<<<<<<<<<<<<<|
        | |  |                  |   Destination IMP
       y++-{ |<<<<<<<<<<<<<<<<<<|
       | | |  __________________
1-10-{ | | | |                  |   Message length (type 0 only)
     \ | | | |                  |   0-9 Padding words (type 0 only)
       | | |
       | | `->|XXXXXXX|_|_._,_|_._,_._._,_._._|
       | |             V \___/ \______________/
       | |             |   V          V
       | |             |   |          `->Host-IMP message type
       | |             |   `->Leader flags (octal print = 2000)
       | |             `->Trace
       | `-->|_,_._._,_._._,_|_._,_._._,_._._|
       |      \______V______/ \______V______/
       |             |               `->Destination Host
       |             `->Handling type
       `--->|_,_._._,_._._,_._._,_._|_,_._._|
             \__________V__________/ \__V__/
                        |               `->Subtype
                        `->Message-ID (Link)
```

IMP to Host

```
                 ______________________
                |XXXX|     |XXXXXXXXX|    New Leader Flag = 7400
            y-{ |<<<<<<<<<<<<<<<<<<<<|
           y+-{ |<<<<<<<<<<<<<<<<<<<<|
            ||  |                    |    Source IMP
          y++-{ |<<<<<<<<<<<<<<<<<<<<|
           |||
 1-10-{    |||  |                    |    Message length (type 0 only)
      \    |||  |                    |    0-9 Padding words (type 0 only)
           |||
           ||`->|XXXXXXXX|_|_._,_|_._,_._._,_._._|
           ||            V  \_V_/  \______V______/
           ||            |    |           `->IMP-Host message type
           ||            |    `->Leader flags
           ||            `->Trace
           |`-->|_,_._._,_._._,_|_._,_._._,_._._|
           |     \______V_______/ \______V______/
           |            |                `->Source Host
           |            `->Handling type
           `--->|_,_._._,_._._,_._._,_._|_,_._._|
                 \__________V__________/ \__V__/
                            |               `->Subtype
                            `->Message-ID (Link) or Host status
```

Buffer format

```
     /      _____________________
4--{   y-{ |<<<<<<<<<<<<<<<<<<<<|  Chain pointer
    |  y+-{|<<<<<<<<<<<<<<<<<<<<|
     \  || |____________________|  INCH: (see below)
        ||  ____________________
8--{    || |____________________|  Packet header words NETH through MIDH
        ||  ____________________
63-{    || |____________________|  Packet data words DATA through DATA+62
        ||  ____________________
1--{    || |____________________|  ITST: input time / sent time
        ||
        |`->|_|_._._,_._._,_._._,_._._,_._._|
        |    V \___________________________/
        |    |              V
        |    |              `->If nonzero, buffer to which this piece
        |    |                   belongs
        |    `->If set, steal this buffer
        |
        `-->|_,_._._,_._._,_|_|_|XXXXXXX|_._|
             \_____________/ V V          \ /
                    V        | |           V
                    |        | |           `->Number of uses of this buffer
                    |        | |                (0=free)
                    |        | `->Set if packet being rerouted
                    |        `->Set if packet being traced
                    `->Header + data word count
```

Format of INCH word
 For Host/modem input

```
    |_|XXXXXXXXXXXXXXXXXXXXXXX|_,_._._|
     V                          \___/
     |                            V
     |                            `->Input channel (0-4 for modem, 0-12
     |                                 for Host)
     `->1 if from Host, 0 if from modem
```

For modem output, INCH first has the logical channel, then a retransmission counter. For Host output, INCH contains the output timeout in slow (640 ms.) ticks.

Basic Packet structure

Not all of the various messages have all of these fields, but if they do have them they will appear as diagrammed here.

```
 ___________________
|                   |  NETH: Modem control bits
|___________________|
|                   |  TYPH: Packet type and other flags
|___________________|
|                   |  CHKH: Software checksum
|___________________|
|                   |  SRCH: Source IMP
|___________________|
|                   |  SEQH: Message/block number
|___________________|
|                   |  PKTH: Packet code, number, and other flags
|___________________|
|                   |  DSTH: Destination IMP
|___________________|
|                   |  MIDH: Message ID
|___________________|
|XXXXXXXXXXXXXXXXXXX|  DATA: Beginning of 0-63 data words
|___________________|
```

Every packet has SRCH, CHKH, and the fields Packet type and Compat.

Packet type is the high order two bits of TYPH.
The basic types are:
0 Data : Message, RFNM, etc
1 Network control: Get-a-block, Reset, etc
2 Switch control: Routing, Null, etc
3 Special: Packet core, Demand reload, etc

Compat is the next bit below packet type and is used as an odd/even bit within packet type so that an incompatible release can be propagated.

For packet types 0 and 1, NETH has the following format:

```
|_|_|XXXXX|_._,_|XXXXXXXXXXXXX|_|
 V V       \___/               V
 | |         V                 |
 | |         |                 `->Always zero
 | |         `->Modem channel number
 | `->HI/LO IMP on line end bit
 `->Odd/even bit for the modem channel
```

Packet type 0: messages concerned with the actual transmission of data

The packet codes are:
 0 Message
 1 Request (for 1 or 8 depending on MLT PKT)
 2 Giveback (Multipacket bit always on)
 3 Incomplete message
 4 RFNM
 5 RFNM w/allocate
 6 Destination was dead
 7 Incomplete reply

Packet format for Type 0, Codes 0-3

```
            ____________________
           |                    |  NETH
     y-{   |<<<<<<<<<<<<<<<<<<<<|
       |   |____________________|  CHKH
       |   |____________________|  SRCH
    y+-{   |<<<<<<<<<<<<<<<<<<<<|
   y++-{   |<<<<<<<<<<<<<<<<<<<<|
   |||     |____________________|  DSTH
  y+++-{   |<<<<<<<<<<<<<<<<<<<<|
  ||||
  |||`->|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
  |||    \_/ V V V \___/ \___________/
  |||     V  | | |   V         V
  |||     |  | | |   |         `->Modem acknowledge bits
  |||     |  | | |   `->Leader flags
  |||     |  | | `->Trace this packet
  |||     |  | `->Priority message
  |||     |  `->Compatibility odd/even bit
  |||     `->Packet type: =0 (Transmission)
  ||`-->|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
  ||     \_____________/ \___________/
  ||           V               V
  ||           |               `->Receive message block number
  ||           `->Message number
  |`--->|_|_|XXX|_|_|_|_|_|_|_|_|_|_|_|_|
  |      V V     \___/ \___/ \___/
  |      | |       V     V     V
  |      | |       |     |     `->Packet code (= 0-3)
  |      | |       |     `->Packet number
  |      | |       `->Receive message block use number
  |      | `->This is the last packet
  |      `->This is multipacket (or part of a multipacket message)
  `---->|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
         \_________________/ \___/
                 V             V
                 |             `->Subtype (only 0 and 3 are
                 |                  defined, see below for =3)
                 `->Message ID
```

Type 0, codes 4-7

```
          _____________________
         |                     |  NETH
    y-{  |<<<<<<<<<<<<<<<<<<<<<|
      |  |                     |  CHKH
      |  |_____________________|  SRCH
   y+-{  |<<<<<<<<<<<<<<<<<<<<<|
  y++-{  |<<<<<<<<<<<<<<<<<<<<<|
   |||   |                     |  DSTH
   |||   |XXXXXXXXXXXXX|       |  Subtype: =0
   |||   |_____________________|  Dead host status
   |||
   |||
   ||`->|_|_|_|_|_|XXXXX|_._._._._._._._|
   ||     \ /  V V V      \_____________/
   ||      V   | | |             V
   ||      |   | | |             `->Modem acknowledge bits
   ||      |   | | `->Trace this packet
   ||      |   | `->Priority (always set)
   ||      |   `->Compatibility odd/even bit
   ||      `->Packet type:  =0 (transmission reply)
   ||
   |`-->|_._._._._._._._|_._._._._._._._|
   |     \_____________/ \_____________/
   |            V               V
   |            |               `->Transmit message block number
   |            `->Message number
   |
   `--->|XXXXXXXXX|_._._._|XXXXX|_._._._|
                   \_____/       \_____/
                      V             V
                      |             `->Packet code (=4-7: means reply)
                      `->Transmit message block use number
```

Type 0, subtype 3: Uncontrolled packet

```
           ____________________
           |                  |   NETH
      y-{  |<<<<<<<<<<<<<<<<<<|
        |  |__________________|   CHKH
        |  |__________________|   SRCH
     y+-{  |<<<<<<<<<<<<<<<<<<|
       ||  |__________________|   Source Host MEMBER access  bits
       ||  |__________________|   DSTH
    y++-{  |<<<<<<<<<<<<<<<<<<|
      |||
      |||  ____________________
      |||  |__________________|   0 to 62 data words
      |||
      |||  ____________________
      |||  |__________________|   Source Host COMMUNICATE access bits
      |||
      ||`->|_'_'_'_'_'_'_'_'_'_'_'_'_'_'_'_|
      ||    \ / V V V \___/ \_____________/
      ||     V  | | |   V          V
      ||     |  | | |   |          `->Modem acknowledge bits
      ||     |  | | |   `->Leader flags
      ||     |  | | `->Trace this packet
      ||     |  | `->Priority message
      ||     |  `->Compatibility odd/even bit
      ||     `->Packet type: =0 (transmission)
      |`-->|_'_'_'_'_'_'_'_'_'_'_'_'_'_'_'_|
      |     \_____________/ \_____________/
      |            V               V
      |            |               `->Source Host
      |            `->Destination Host
      `--->|_'_'_'_'_'_'_'_'_'_'_'_'_'_'_'_|
            \___________________/ \_____/
                      V              V
                      |              `->Sub type: =3 (marks as
                      |                   uncontrolled)
                      `->Message ID
```

The type 1 packets use the following codes:
  10 Incomplete query
  11 Get-a-block
  12 Reset block
  13 .. unused
  14 Out of range
  15 Got-a-block or Got-no-block (i.e., reply to a get-a-block)
  16 Reset block request
  17 Reset block reply
Packet type 1, codes 10, 12, 14, 15, 16 and 17

```
             ____________________
            |____________________|  NETH
       y-{  |<<<<<<<<<<<<<<<<<<<<|
        |   |____________________|  CHKH
        |   |____________________|  SRCH
      y+-{  |<<<<<<<<<<<<<<<<<<<<|
     y++-{  |<<<<<<<<<<<<<<<<<<<<|
      |||   |____________________|  DSTH
    y+++-{  |<<<<<<<<<<<<<<<<<<<<|
     ||||   |____________________|  Dead host status
     ||||
     |||`->|_|_|_|_|_|_|_|_|_._._._._._._._|
     |||     \ / V V V V V V \_____________/
     |||      V  | | | | | |        V
     |||      |  | | | | | |        `->Modem acknowledge bits
     |||      |  | | | | | `->Host dead or non-existent (code 15)
     |||      |  | | | | `->Host access violation (code 15)
     |||      |  | | | `->Used an allocate (code 10)
     |||      |  | | `->Trace this packet
     |||      |  | `->Priority (always set)
     |||      |  `->Compatibility odd/even bit
     |||      `->Packet type: =1 (net control)
     |||
     ||`-->|_._._._._._._._|_._._._._._._._|
     ||     \_____________/ \_____________/
     ||            V               V
     ||            |               `->Destination message block number
     ||            `->Message number (codes 10 & 14)
     ||
     |`--->|XXXXXXX|_._._._|XXXXXXX|_._._._|
     |              \_____/         \_____/
     |                 V               V
     |                 |               `->Packet code: =10,12,14-17
     |                 `->Destination message block use number
     |
     `---->|XXXXXXX|_._._._|_._._._._._._._|
                    \_____/ \_____________/
                       V           V
                       |           `->Source message block number
                       `->Source message block use number
```

Packet type 1, code 11

```
              ___________________
             |                   |  NETH
      y-{    |<<<<<<<<<<<<<<<<<<<|
        |    |                   |  CHKH
        |    |___________________|  SRCH
     y+-{    |<<<<<<<<<<<<<<<<<<<|
       ||    |XXXXXXXXXXXXX|_____|  Packet code: =11 (get-a-block)
       ||    |___________________|  DSTH
    y++-{    |<<<<<<<<<<<<<<<<<<<|
     |||     |                   |  Source Host MEMBER access bits
     |||     |___________________|  Source Host COMMUNICATE access bits
     |||     |___________________|
     |||
     ||`->|_,_|_|_|_|_|XXX|_._,_._._,_._._|
     ||    \ /  V V V V    \_____________/
     ||     V   | | | |           V
     ||     |   | | | |           `->Modem acknowledge bits
     ||     |   | | | `->Need an allocate
     ||     |   | | `->Trace this packet
     ||     |   | `->Priority (always set)
     ||     |   `->Compatibility odd/even bit
     ||     `->Packet type: =1 (net control)
     |`-->|_,_._._,_._._,_|_._,_._._,_._._|
     |     \_____________/ \_____________/
     |            V               V
     |            |               `->Source Host
     |            `->Destination Host
     `--->|_,_._._|_._._,_|_._,_._._,_._._|
           \_____/ \_____/ \_____________/
              V       V           V
              |       |           `->Source message block number
              |       `->Source message block use number
              `->Handling type
```

Packet type 2: Routing and null

Routing packet

```
           y-{|<<<<<<<<<<<<<<<<<<<<|
          y+-{|<<<<<<<<<<<<<<<<<<<<|
           ||  |                    |  CHKH
           ||  |____________________|  SRCH
           ||
67---{     ||  |                    |  Routing words for IMPs 1 to 67
           ||
           |`->|_,_._._,_._._,_|XXXXXXXXXXXXXX|_|
           |    \_____________/               V
           |          V                       |
           |          |                       `->=0
           |          `->Serial number
           |
           `-->|_,_|_|XXX|_._|_|XXXXXXXXXXXXXXX|
                \ / V     \ / V
                 V  |      V  |
                 |  |      |  `->=0 (marks as ROUTING)
                 |  |      `->Compatibility number
                 |  `->Compatibility odd/even bit
                 `->Packet type: =2 (routing type)
```

Format for word N of routing data in above message

```
            |_,_._._,_|_._,_._._,_._._,_._._|
             \_______/ \___________________/
                 V               V
                 |               `->Delay to IMP N
                 `->Number of hops to IMP N
```

Null packet

```
              |XXXXXXXXXXXXXXXX|  |  =0
          y-{|<<<<<<<<<<<<<<<<<<<<|
           |  |                    |  CHKH
           |  |                    |  SRCH
           |  |____________________|  Sync time
           |
           `->|_,_|_|XXX|_|_|_|_._,_._._,_._._|
               \ / V     V V V \______________/
                V  |     | | |        V
                |  |     | | |        `->Modem acknowledge bits
                |  |     | | `->=1 (mark as NULL)
                |  |     | `->"I heard you"
                |  |     `->"I am a stub"
                |  `->Compatibility odd/even bit
                `->Packet type: =2 (routing type)
```

Packet type 3: Demand reload, Reload request and Packet core

Demand reload

```
       _________________________
      |XXXXXXXXXXXXXXXXX|   |    =0
  y-{ |<<<<<<<<<<<<<<<<<<<<<|
   |  |_____________________|    CHKH
   |  |_____________________|    Reload code
   |  |_____________________|    Password
   |
   |
   `->|_,_|_|XXX|_._,_|XXXXXXXXXXXXXXXX|
       \ /  V      \___/
        V   |        V
        |   |        `->=1
        |   `->Compatibility odd/even bit
        `->Packet type: =3 (special)
```

Reload request

```
      |XXXXXXXXXXXXXXXXX|   |    =0
  y-{ |<<<<<<<<<<<<<<<<<<<<<|
   |  |_____________________|    CHKH
   |
   |
   `->|_,_|_|XXX|_._,_|XXXXXXXXXXXXXXXX|
       \ /  V      \___/
        V   |        V
        |   |        `->=2
        |   `->compatibility odd/even bit
        `->Packet type: =3 (special)
```

Packet core

```
       y-{|<<<<<<<<<<<<<<<<<<<<|
      y+-{|<<<<<<<<<<<<<<<<<<<<|
       ||  |____________________|  CHKH
       ||  |____________________|  Satellite source IMP
       ||  |____________________|  Satellite destination IMP
     y++-{|<<<<<<<<<<<<<<<<<<<<|
      |||  |____________________|  DSTH
    y+++-{|<<<<<<<<<<<<<<<<<<<<|
    ||||
    |||`->|_'_'_'_'_'_'_'_'_|_'_'_'_'_'_'_'_|
    |||    \_______________/ \_____________/
    |||            V                V
    |||            |                `->Packet size (must be even)
    |||            `->=0
    ||`-->|_'_|_|XXX|_'_'_|XXXXXXXXXXXXXXXXX|
    ||     \ / V     \___/
    ||      V  |       V
    ||      |  |       `->=0
    ||      |  `->Compatibility odd/even bit
    ||      `->Packet type: =3 (special)
    |`--->|_'_'_'_'_'_'_'_'_|_'_'_'_'_'_'_'_|
    |      \_______________/ \_____________/
    |              V                V
    |              |                `->Destination Host
    |              `->Handling type
    `---->|_'_'_'_'_'_'_'_'_'_'_'_|_'_'_'_|
           \_____________________/ \_____/
                      V               V
                      |               `->Subtype
                      `->Message ID
```

Data for SETUP message

```
        ____________________
    y-{|<<<<<<<<<<<<<<<<<<<|
   y+-{|<<<<<<<<<<<<<<<<<<<|
    || |___________________|  Foreign IMP
  y++-{|<<<<<<<<<<<<<<<<<<<|
 y+++-{|<<<<<<<<<<<<<<<<<<<|
 ||||  |___________________|  Starting address
 ||||  |___________________|  Transfer size
 ||||  |___________________|  Send setup flag
 ||||  |___________________|  Send/Receive flag
 ||||
 |||`->;_,_._._,_._._;_;_._,_._._,_._;
 |||   ;_,_._._,_._._;_;_._,_._._,_._;
 |||    \___________/ V \___________/
 |||          V       |       V
 |||          |       |       `->Destination line number
 |||          |       `->Magic modem bit
 |||          `->"SETUP" code
 ||`-->;_,_._._,_._._,_;_._,_._._,_._;
 ||    ;_,_._._,_._._,_;_._,_._._,_._;
 ||     \_____________/ \___________/
 ||           V               V
 ||           |               `->Foreign Host number
 ||           `->Foreign handling type
 |`--->;_,_._._,_._._,_._._,_._;_,_._;
 |     ;_,_._._,_._._,_._._,_._;_,_._;
 |      \___________________/ \___/
 |                V              V
 |                |              `->Subtype
 |                `->Foreign message ID
 `---->;_,_._._,_._._;_;_._,_._._,_._;
       ;_,_._._,_._._;_;_._,_._._,_._;
        \___________/ V \___________/
              V       |       V
              |       |       `->Foreign Line number
              |       `->Magic modem bit
              `->Core type
```

Data for CORE message

```
                  y-{|<<<<<<<<<<<<<<<<<<<<<|
                    |
            /     y+-{|<<<<<<<<<<<<<<<<<<<<<|
Core pieces-{   y++-{|<<<<<<<<<<<<<<<<<<<<<|
            \    |||  |                     |   The words of data
                 |||
                 ||`->|_,_._,_._,_._,_|_|_._,_._,_._,_|
                 ||   \_______________/ V \______________/
                 ||           V         |        V
                 ||           |         |        `->Destination line number
                 ||           |         `->Magic modem bit
                 ||           `->"CORE" code
                 |`-->|_,_._,_._,_._,_._,_._,_._,_._,_|
                 |    \_______________________________/
                 |                     V
                 |                     `->Starting address of this piece
                 |                          (or 0 if no more)
                 `--->|_,_._,_._,_._,_|_._,_._,_._,_._|
                      \_______________/\_______________/
                              V                V
                              |                `->Size of this piece
                              `->Core type
```

Trace Block Table - parallel table 11 words wide

```
            ____________________
           |                    |  TPT: Pointer to packet being traced
           |____________________|  TIT: Input time
           |____________________|  TTT: Task time
           |____________________|  TST: Sent time
           |____________________|  TAT: Ack time
           |____________________|  Source IMP
       y-{ |<<<<<<<<<<<<<<<<<<<<|
      y+-{ |<<<<<<<<<<<<<<<<<<<<|
       ||  |____________________|  Destination IMP
     y++-{ |<<<<<<<<<<<<<<<<<<<<|
    y+++-{ |<<<<<<<<<<<<<<<<<<<<|
    ||||
    |||`-> |_'_'_'_'_'_'_'_|_'_'_'_'_'_'_'_|
    |||     \_____________/ \_____________/
    |||            V               V
    |||            |               `->Destination message block number
    |||            `->Message number
    ||`--> |_|_|X|_|_|_'_'_|_'_'_|_'_'_|
    ||      V V   V V \___/ \___/ \___/
    ||      | |   | |   V     V     V
    ||      | |   | |   |     |     `->Packet code
    ||      | |   | |   |     `->Packet number
    ||      | |   | |   `->Source flags
    ||      | |   | `->Source requested trace
    ||      | |   `->Priority
    ||      | `->Last packet
    ||      `->Multi-packet
    |`---> |_'_'_'_'_'_'_'_'_'_'_'_|_'_'_'_|
    |       \_____________________/ \_____/
    |                 V                V
    |                 |                `->Subtype
    |                 `->Message-ID
    `----> |_|_|_'_'_'_'_'_|_'_'_|_'_'_|
            V V \_______/ \___/ \___/
            | |     V       V     V
            | |     |       |     `->Output channel
            | |     |       `->Input channel
            | |     `->Packet size (not including 8 header words)
            | `->Packet rerouted
            `->Tracing done
```

Transmit Message Block Table - parallel table 5 words wide

```
              ____________________
             |                    |  Receive IMP (0 if block is unused)
         y-{ |<<<<<<<<<<<<<<<<<<<<|
        y+-{ |<<<<<<<<<<<<<<<<<<<<|
       y++-{ |<<<<<<<<<<<<<<<<<<<<|
      y+++-{ |<<<<<<<<<<<<<<<<<<<<|
      ||||
      |||`->|_'_'_'_'_'_'_'_|_'_'_'_'_'_'_'_|
      |||    \______________/ \______________/
      |||           V                 V
      |||           |                 `->Transmit Host
      |||           `->Receive Host
      ||`-->|_'_'_'_|_'_'_'_|_'_'_'_'_'_'_'_|
      ||     \_____/ \_____/ \______________/
      ||        V       V           V
      ||        |       |           `->Receive message block number
      ||        |       `->Receive message block use number
      ||        `->Handling type
      |`--->|_'_'_'_'_'_'_'_|_'_'_'_|_'_'_'_|
      |      \______________/ \_____/ \_____/
      |             V            V       V
      |             |            |       `->Age
      |             |            `->Transmit message block use number
      |             `->TMESS- Next message number to send
      `---->|_|_|_|_'_'_|_'_|_'_'_'_'_'_'_'_|
             V V V \___/ \_/ \______________/
             | | |   V    V         V
             | | |   |    |         `->Outstanding messages: bit i (1 to 8,
             | | |   |    |              left to right) =1 means message
             | | |   |    |              TMESS-i is complete
             | | |   |    `->Incomplete query, get-a-block, and reset
             | | |   |          timeout
             | | |   `->Allocate count
             | | `->Clear messge pipe
             | `->Initialize
             `->Reset
```

Receive Message Block Table - parallel table 6 words wide

```
            ____________________
            |                  |  Transmit IMP (0 if block is unused)
     y-{    |<<<<<<<<<<<<<<<<<<|
    y+-{    |<<<<<<<<<<<<<<<<<<|
   y++-{    |<<<<<<<<<<<<<<<<<<|
   |||      |__________________|  2-bit reply states for 8 messages (see
   |||      |__________________|  2-bit reply types for 8 messages (see
   |||
   ||`->;_,_._._,_._._,_;_._,_._._,_._._;
   ||   ;_,_._._,_._._,_;_._,_._._,_._._;
   ||    \_____________/ \_____________/
   ||           V               V
   ||           |               `->Receive Host
   ||           `->Transmit Host
   ||
   |`-->;_,_._._;_._._,_;_._,_._._,_._._;
   |    ;_,_._._;_._._,_;_._,_._._,_._._;
   |     \_____/ \_____/ \_____________/
   |        V       V           V
   |        |       |           `->Transmit message block number
   |        |       `->Transmit message block use number
   |        `->Handling type
   |
   `--->;_,_._._,_._._,_;_._,_._;_,_._._;
        ;_,_._._,_._._,_;_._,_._;_,_._._;
         \_____________/ \_____/ \_____/
                V           V       V
                |           |       `->Age
                |           `->Receive message block use number
                `->RMESS- Next message number to give to Host
```

State and type words contain 8 2-bit entries, numbered i=1,...,8 (left to right), where entry number i corresponds to either message number RMESS-i (previous window, *P*), or to message number RMESS+8-i (current window, *C*), depending on the values of state and type. The meanings of various combinations, along with the window (*P* or *C*), is given by the following table:

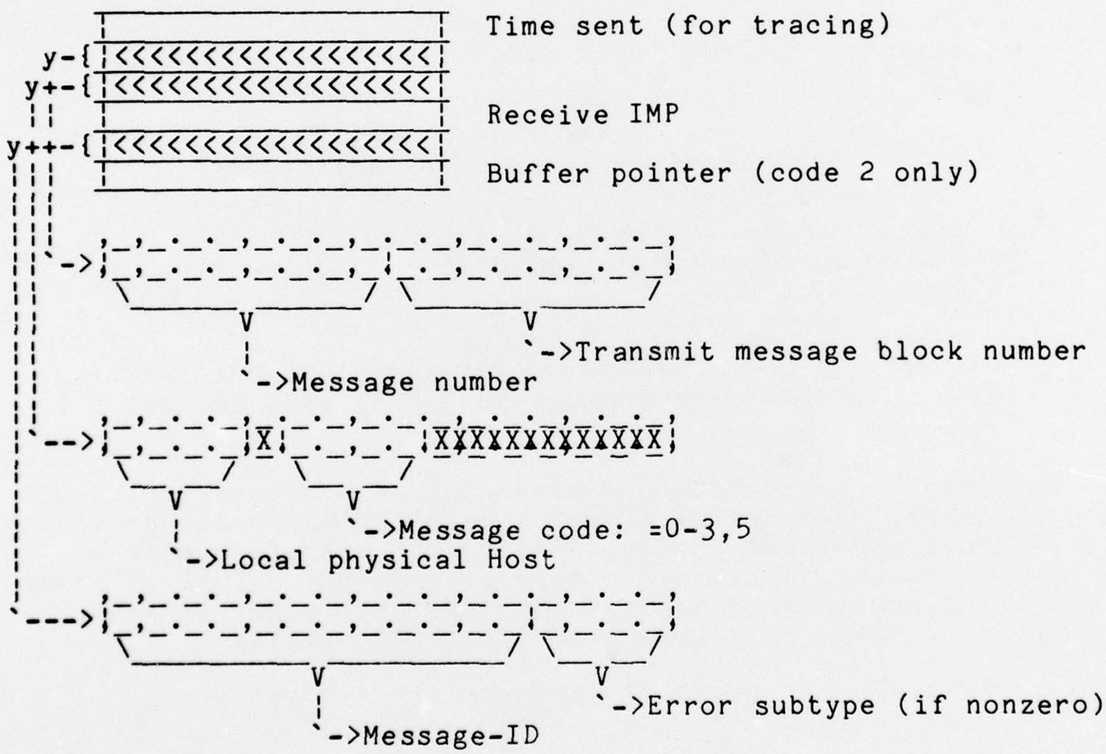
STATE-> TYPE ! V	IDLE 00	REQUEST 01	MESSAGE 10	REPLY 11
00	RFNM SENT *P*	REQ1 RCVD *C*	ALL1 SENT/ MSG RCVD *C*,*P*	RFNM1 TO BE SENT *P*
01	ALL8 SENT *P*	REQ8 RCVD *C*	GVB RCVD *C*	RFNM8 TO BE SENT *P*
10	DEAD SENT *P*	ALL1 TO BE SENT *C*	DEAD RCVD *C*	DEAD TO BE SENT *P*
11	INC SENT *P*	ALL8 TO BE SENT *P*	INC RCVD *C*	INC TO BE SENT *P*

Transaction Block Table - parallel table 6 words wide

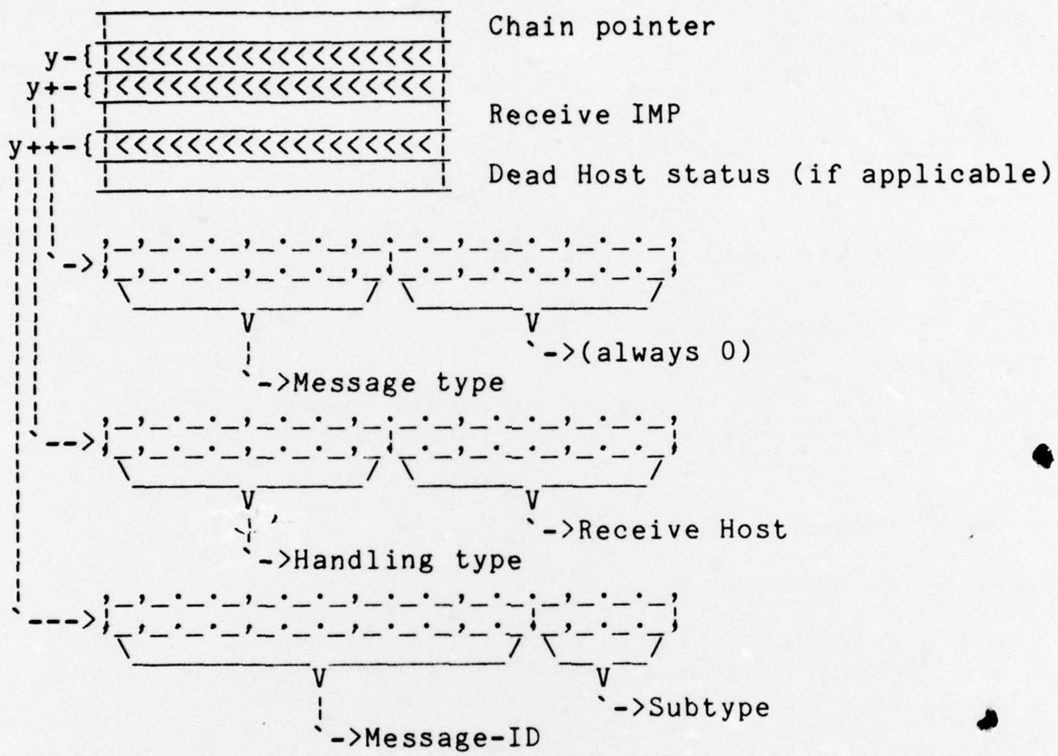
The second word is zero for unused blocks.

Transaction Block Format for Outstanding Messages

The outstanding message codes are:
- 0 Single-packet message
- 1 Multi-packet message
- 2 Single-packet request
- 3 Multi-packet request
- 5 Giveback



Transaction Block Format for Control Message Queue



Reassembly Block Table - parallel table 3 words wide

```
         ______________________
        |                      |  Packet chain pointer
  y-{   |<<<<<<<<<<<<<<<<<<<<<<|
 y+-{   |<<<<<<<<<<<<<<<<<<<<<<|
  | |    ----------------------
  | |
  | `->  _______________________________________
  |     |___________________|___________________|
  |      \_________________/ \_________________/
  |              V                    V
  |              |                    `->Receive message block number
  |              `->Message number
  |
  `-->   _______________________________________
        |___________________|___________________|
         \_________________/ \_________________/
                 V                    V
                 |                    `->RSF: number of packets so far
                 `->RMAX: number of packets in message
```

Reassembly block states:

Unused - all three words zero.

No-name - pointer to SIGN-PKTH in first word, number of packets allocated-1/receive message block number+200 in second word, third word zero.

Partial - chain pointer in first word, message/block numbers in second word, and in third word either (before last packet received) 16+number allocated-1/number received-1, or (after last packet received) last packet number/number of packets received-1.

Complete - chain pointer in first word, message/block numbers in second word, and -number of packets received in third word.

RMFLG, Routing Message Flag Table - 1 word/modem

```
     ,_,_._._,_._._,_._._._,_._._,_._._,
     |_|_._._,_._._,_|XXXXX|_|_|_._|_|
      V \___________/       V V \ / V
      |       V             | |  V  |
      |       |             | |  |  `->Routing message pending
      |       |             | |  `->Speed hold-down counter
      |       |             | `->Packet Core message pending
      |       |             `->Line reset pending
      |       `->0=5Kbs, 1=10Kbs, 2=50Kbs, 3=250Kbs nominal line speed
      `->Set in init to suppress trap on load
```

RUT, Route Use Table, best line directory - 1 word/IMP

```
     ,_,_._._,_._._._,_._._,_._._,_._._,
     |_|_._|_,_._._._,_|_._,_|_._,_._._|
      V \ / \_______/ \___/ \_______/
      |  V      V       V       V
      |  |      |       |       `->Line #+1 of shortest delay path
      |  |      |       `->Coming up delay counter
      |  |      `->Line #+1 of shortest hop path
      |  `->Going down delay counter
      `->IMP down, unreachable or isolated
```

RUTW, RUT Working Table - 1 word/IMP

```
     ,_,_._._,_._._._,_._._,_._._,_._._,
     |_|_._|_,_._._._,_._|_,_|_._,_._._|
      V \ / \_________/ \ / \_______/
      |  V      V        V      V
      |  |      |        |      `->0
      |  |      |        `->Hold-down timer for minimum delay path
      |  |      `->0
      |  `->Hold-down timer for minimum hop path
      `->0
```

REFERENCES

1. Heart, F.E., Kahn, R.E., Ornstein, S.M., Crowther, W.R., and Walden, D.C., "The Interface Message Processor for the ARPA Computer Network." Proceedings of AFIPS 1970 Spring Joint Computer Conference, Vol. 36, pp. 551-567.

2. McQuillan, J.M., Crowther, W.R., Cosell, B.P., Walden, D.C., Heart, F.E., "Improvements in the Design and Performance of the ARPA Network." Proceedings of AFIPS 1972 Fall Joint Computer Conference, Vol. 41, pp. 741-754.

3. Bolt Beranek and Newman, Inc. "Specification for the Interconnection of a Host and an IMP." Report No. 1822, revised December 1974.

4. Bobrow, D.G., Burchfield, J.D., Murphy, D.C., and Tomlinson, R.S., "TENEX, a Paged Time-sharing System for the PDP-10." Communications of the ACM, Vol. 15, No. 3, March 1972, pp. 135-143.